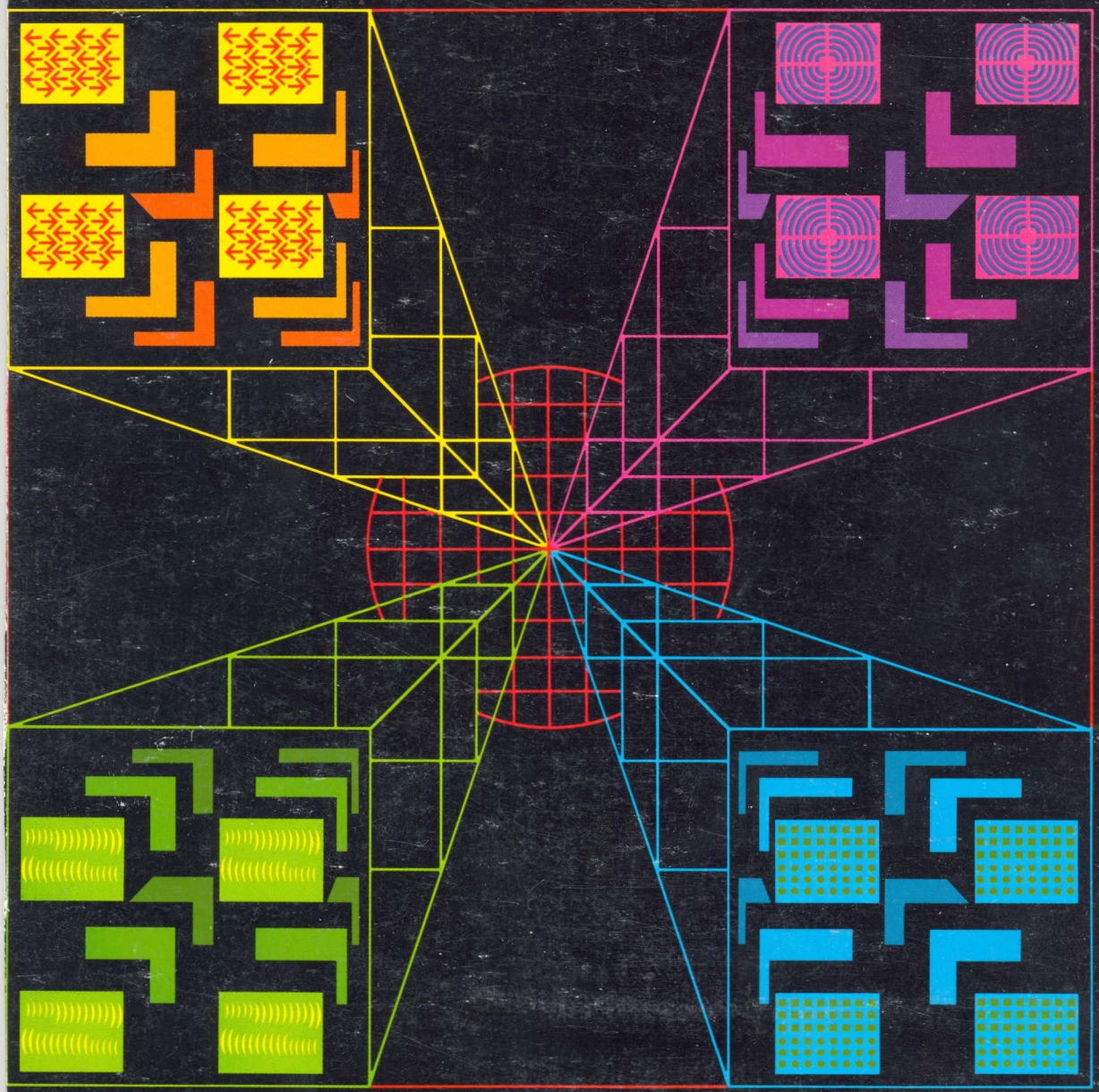
intel

MCS®-96 User's Manual



Order Number: 230883-001

LITERATURE

In addition to the product line Handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846) provides an overview of Intel's complete product line and customer services.

Consult the INTEL LITERATURE GUIDE for a complete listing of Intel literature. TO ORDER literature in the United States, write or call the Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only). TO ORDER literature from international locations, contact the nearest Intel sales office or distributor (see listings in the back of most any Intel literature).

1984 HANDBOOKS	U.S. PRICE*
Memory Components Handbook (Order No. 210830) Contains all application notes, article reprints, data sheets, and other design information on RAMs, DRAMs, EPROMs, E^2PROMs, Bubble Memories.	**$15.00**
Telecommunication Products Handbook (Order No. 230730) Contains all application notes, article reprints, and data sheets for telecommunication products.	**7.50**
Microcontroller Handbook (Order No. 210918) Contains all application notes, article reprints, data sheets, and design information for the MCS-48, MCS-51 and MCS-96 families.	**15.00**
Microsystem Components Handbook (Order No. 230843) Contains application notes, article reprints, data sheets, technical papers for microprocessors and peripherals. (2 Volumes) (Individual User Manuals are also available on the 8085, 8086, 8088, 186, 286, etc. Consult the Literature Guide for prices and order numbers.)	**20.00**
Military Handbook (Order No. 210461) Contains complete data sheets for all military products. Information on Leadless Chip Carriers and on Quality Assurance is also included.	**10.00**
Development Systems Handbook (Order No. 210940) Contains data sheets on development systems and software, support options, and design kits.	**10.00**
OEM Systems Handbook (Order No. 210941) Contains all data sheets, application notes, and article reprints for OEM boards and systems.	**15.00**
Software Handbook (Order No. 230786) Contains all data sheets, applications notes, and article reprints available directly from Intel, as well as 3rd Party software.	**10.00**

*** Prices are for the U.S. only.**

intel

MCS®-96 USER'S MANUAL

1984

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i, $\hat{\text{i}}$, ICE, iCS, iDBP, iDIS, I^2ICE, iLBX, i_m, iMMX, Insite, Intel, int$_e$l, int$_e$lBOS, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPDS, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, Plug-A-Bubble, PROMPT, Promware, QUEST, QUEX, Ripplemode, RMX/80, RUPI, Seamless, SOLO, SYSTEM 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

Introduction to MCS®-96

1

CHAPTER 1
INTRODUCTION TO MCS®-96

1.0 CONTINUING MICROCONTROLLER EVOLUTION

Beginning with the introduction of the world standard 8048 (MCS®-48) Microcontroller in 1976, Intel has continued to drive the evolution of single chip microcontrollers. In 1980, Intel introduced the 8051 (MCS-51) offering performance levels significantly higher than the 8048. With the advent of the 8051, the microcontroller applications base took a marked vertical leap. These versatile chips are used in applications from keyboards and terminals to controlling automobile engines. The 8051 quickly gained the position of the second generation world standard microcontroller.

Now that the semiconductor process technologies are being pushed to new limits, it has become possible to integrate more than 100,000 transistors onto a single silicon chip. Microcontroller designers at Intel have taken today's process technology achievements and forged a new generation of single chip microcontrollers called the MCS-96. The 8096 (generic part number for MCS-96) offers the highest level of system integration ever achieved on a single chip microcontroller. It uses over 120,000 transistors to implement a high performance 16-bit CPU, 8K bytes of program memory, 232 bytes of data memory and both analog and digital types of I/O features. Figure 1-1 shows the evolution of single chip microcontroller at Intel.

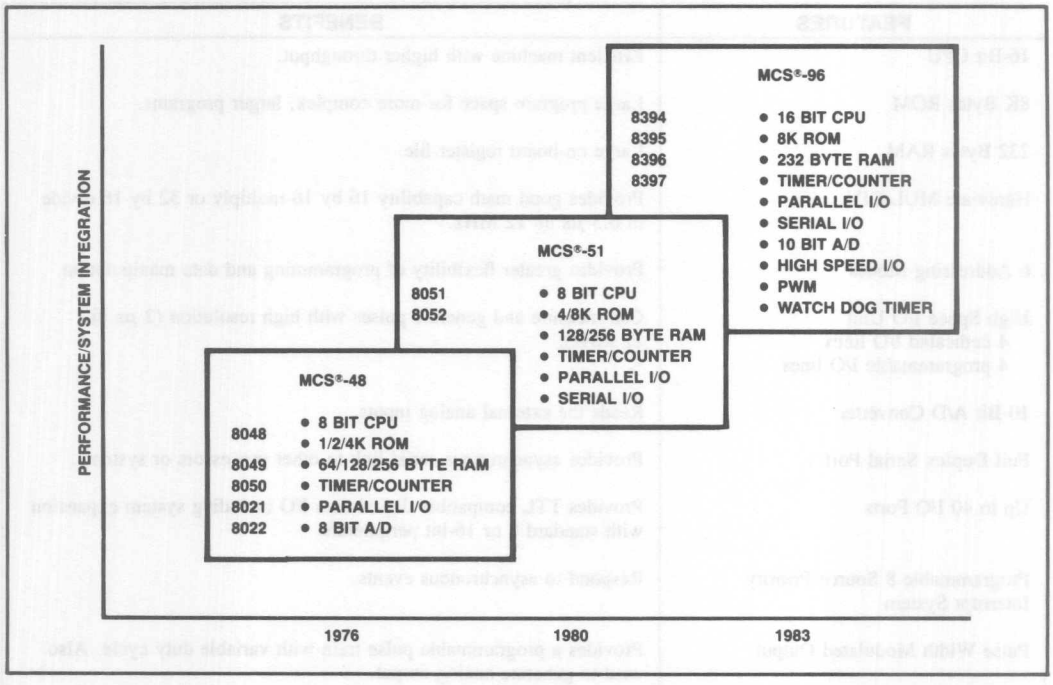


Figure 1-1. Evolution of Microcontrollers at Intel

1.1 INTRODUCTION TO THE MCS®-96

The 8096 consists of a 16-bit powerful CPU tightly coupled with program and data memory along with several I/O features all integrated onto a single piece of silicon. The CPU supports bit, byte, and word operations. 32-bit double words are also supported for a subset of the instruction set. With a 12 MHz input frequency, the 8096 can perform a 16-bit addition in 1.0 μs and 16 $\times$ 16 multiply or 32/16 divide in 6.5 μs.

Four high-speed trigger inputs are provided to record the times at which external events occur with a resolution of 2 μs (at 12 MHz crystal frequency). Up to six high-speed pulse generator outputs are provided to trigger external events at preset times. The high speed output unit can simultaneously perform timer functions, up to four such 16-bit software timers can be in operation at once in addition to the two 16-bit hardware timers.

An optional on-chip A/D converter converts up to four (in the 48-pin version) or 8 (in the 68-pin version) analog input channels into 10-bit digital values. Also provided on-chip, is a serial port, a watchdog timer, and a pulse-width modulated output signal. Table 1.1 shows the features and benefits summary for the MCS-96.

The 8096 with its 16-bit CPU and all the I/O features and interface resources on a single piece of silicon represents the highest level of system integration in the world of microcontrollers. It will open up new applications which had to use multiple chip solutions in the past.

Table 1-1 MCS®-96 Features and Benefits Summary

FEATURES	BENEFITS
16-Bit CPU	Efficient machine with higher throughput.
8K Bytes ROM	Large program space for more complex, larger programs.
232 Bytes RAM	Large on-board register file.
Hardware MUL/DIV	Provides good math capability 16 by 16 multiply or 32 by 16 divide in 6.5 μs @ 12 MHz.
6 Addressing Modes	Provides greater flexibility of programming and data manipulation.
High Speed I/O Unit 4 dedicated I/O lines 4 programmable I/O lines	Can measure and generate pulses with high resolution (2 μs @ 12 MHz).
10-Bit A/D Converter	Reads the external analog inputs.
Full Duplex Serial Port	Provides asynchronous serial link to other processors or systems.
Up to 40 I/O Ports	Provides TTL compatible digital data I/O including system expansion with standard 8 or 16-bit peripherals.
Programmable 8 Source Priority Interrupt System	Respond to asynchronous events.
Pulse Width Modulated Output	Provides a programmable pulse train with variable duty cycle. Also used to generate analog output.
Watchdog Timer	Provides ability to recover from software malfunction or hardware upset.
48 Pin (DIP) & 68 Pin (Flatpack, Pin Grid Array) Versions	Offers a variety of package types to choose from to better fit a specific application need for number of I/O's and package size.

1.2. MCS®-96 APPLICATIONS

The MCS-96 products are stand-alone high performance single chip microcontrollers designed for use in sophisticated real-time demanding applications such as industrial control, instrumentation and intelligent computer peripherals. The wide base of applications cut across all industry segments (see table 1.2). With the 16-bit CPU horsepower, high-speed math processing and high-speed I/O, the 8096 is ideal for complex motor control and axis control systems. Examples include three phase, large horsepower AC motors and robotics.

With its 10-bit A/D converter option, the device finds usage in data acquisition systems and closed-loop analog controllers. It permits considerable system integration by combining analog and digital I/O processing in the single chip.

This chip is ideally suited in the area of instrumentation products such as gas chromatographs, which combine analog processing with high speed number crunching. The same features make it a desirable component for aerospace applications like missile guidance and control.

Table 1-2 MCS®-96 Broad Base of Applications

INDUSTRIAL
Motor Control
Robotics
Discrete and Continuous Process Control
Numerical Control
Intelligent Transducers
INSTRUMENTATION
Medical Instrumentation
Liquid and Gas Chromatographs
Oscillioscopes
CONSUMER
Video Recorder
Laser Disk Drive
High-end Video Games
GUIDANCE & CONTROL
Missile Control
Torpedo Guidance Control
Intelligent Ammunition
Aerospace Guidance Systems
DATA PROCESSING
Plotters
Color and B&W Copiers
Winchester Disk Drive
Tape Drives
Impact and Non-Impact Printers
TELECOMMUNICATIONS
Modems
Intelligent Line Card Control
AUTOMOTIVE
Ignition Control
Transmission Control
Anti Skid Braking
Emission Control

1.3. MCS®-96 FAMILY DEVELOPMENT SUPPORT TOOLS

The product family is supported by a range of Intel software and hardware development tools. These tools shorten the product development cycle, thus bringing the product to the market sooner.

1.3.1. MCS®-96 Software Development Package

The 8096 software development package provides development system support specifically designed for the MCS-96 family of single chip microcontrollers. The package consists of a symbolic macro assembler ASM-96, Linker/Relocator RL-96 and the librarian LIB-96. Among the high level languages, PLM-96 is offered along with a floating point math package. Additional high level languages are being developed for the MCS-96 product family.

1.3.2. ASM-96 MACRO Assembler

The 8096 macro assembler translates the symbolic assembly language instructions into the machine executable object code. ASM-96 enables the programmer to write the program in a modular fashion. The modular programs divide a rather complex program into smaller functional units, that are easier to code, to debug, and to change. The separate modules can then be linked and located into one program module using the RL-96 utility. This utility combines the selected input object modules into a single output object module. It also allocates memory to input segments and binds the relocatable addresses to absolute addresses. It then produces a print file that consists of a link summary, a symbol table listing and an intermediate cross-reference listing. LIB-96, another utility helps to create, modify, and examine library files. The ASM-96 runs on Intellec Series III or IV.

1.3.3. PL/M-96

The PL/M-96 compiler translates the PL/M-96 language into 8096 relocatable object modules. This allows improved programmer productivity and application reliability. This high level language has been efficiently designed to map into the machine architecture, so as not to trade off higher programmer productivity with inefficient code. Since the language and the compiler are optimized for the 8096 and its application environment, developing software with PL/M-96 is a 'low-risk' project.

1.3.4. Hardware Development Support: iSBE-96

The iSBE-96 is a hardware execution and debug tool for the MCS-96 products. It consists of a monitor/debugger resident in an 8096 system. This development system interfaces with the user's 8096 system via two ribbon cables, one for the 8096 I/O ports, and the other for the memory bus. The iSBE-96 is controlled by an Intellec Series III or other computer system over a serial link. Power for the iSBE-96 can be supplied by plugging it into the MULTIBUS® card slot, or by an external power supply. The iSBE-96 is contained on one standard MULTIBUS board.

The iSBE-96 provides the most often used features for real-time hardware emulation. The user can display and modify memory, set up break points, execute with or without breakpoints and change the memory map. In addition, the user can single step through the system program.

1.3.5. MCS®-96 Workshop

The workshop provides the design engineer or system designer hands-on experience with the MCS-96 family of products. The course includes an explanation of the Intel 8096 architecture, system timing, input/output design. The lab sessions allow the attendees to gain in-depth knowledge of the MCS-96 product family and support tools.

1.3.6. Insite™ Library

The Intel Insite Library contains several application programs. A very useful program contained in the Insite is SIM-96, the software simulator for 8096. It allows software simulations of user's system. The simulator provides the ability to set breakpoints, examine and modify memory, disassemble the object code and single step through the code.

1.4. MCS®-96 FAMILY OF PRODUCTS

Although 8096 is the generic part number often used for the MCS-96 products throughout this manual, the product family consists of eight configurations with eight part numbers including the 8096. This wide variety of products is offered to best meet user's application requirements in terms of number of I/O's and package size. The options include on-board 8K bytes of mask programmed memory, 10-bit A/D converter, and 48 or 68 pin package type.

Table 1-3 summarizes all the current products in the MCS®-96 product family.

Table 1-3 MCS®-96 Family of Products

OPTIONS		68 PIN	48 PIN
DIGITAL I/O	ROMLESS	8096	8094
	ROM	8396	8394
ANALOG AND DIGITAL I/O	ROMLESS	8097	8095
	ROM	8397	8395

The 48 pin version is available in a DIP (dual inline) package.

The 68 pin version comes in two packages, the Plastic Flatpack and the Pin Grid Array.

Architectural Overview 2

CHAPTER 2
ARCHITECTURAL OVERVIEW

2.0. INTRODUCTION

The 8096 can be separated into several sections for the purpose of describing its operation. There is a CPU, a programmable High Speed I/O Unit, an analog to digital converter, a serial port, and a Pulse Width Modulated (PWM) output for digital to analog conversion. In addition to these functional units, there are some sections which support overall operation of the chip such as the clock generator and the back-bias generator. The CPU and the programmable I/O make the 8096 very different from any other microcontroller, let us first examine the CPU.

2.1. CPU OPERATION

The major components of the CPU on the 8096 are the Register File and the RALU. Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU (Register/Arithmetic Logic Unit) does not use an accumulator, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, the absence of accumulator bottleneck, and fast throughput and I/O times.

2.1.1. CPU Buses

A "Control Unit" and two buses connect the Register File and RALU. Figure 2-1 shows the CPU with its major bus connections. The two buses are the "A-Bus" which is 8-bits wide, and the "D-Bus" which is 16-bits wide. The D-Bus transfers data only between the RALU and the Register File or Special Function Registers (SFRs). The A-Bus is used as the address bus for the above transfers or as a multiplexed address/data bus connecting to the "Memory Controller". Any accesses of either the internal ROM or external memory are done through the Memory Controller.

Within the memory controller is a slave program counter (Slave PC) which keeps track of the PC in the CPU. By having most program fetches from memory referenced to

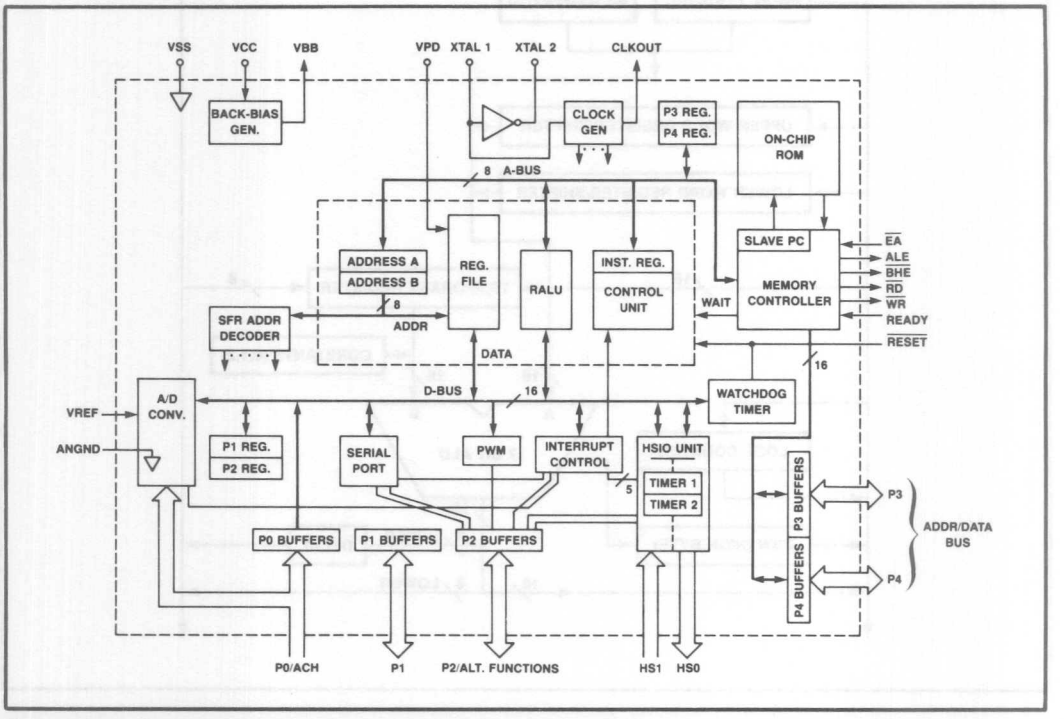


Figure 2-1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)

the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address jumps sequence then the slave PC is loaded with a new value and processing continues. Data fetches from memory are also done through the memory controller, but the slave PC is bypassed for this operation.

2.1.2. CPU Register File

The Register File contains 232 bytes of RAM which can be accessed as bytes, words, or double-words. Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". The first word in the Register File is reserved for use as the stack pointer so it can not be used for data when stack manipulations are taking place. Addresses for accessing the Register File and SFRs are temporarily stored in two 8-bit address registers by the CPU hardware.

2.1.3. RALU Control

Instructions to the RALU are taken from the A-Bus and stored temporarily in the instruction register. The Control Unit decodes the instructions and generates the correct sequence of signals to have the RALU perform the desired function. Figure 2-1 shows the instruction register and the control unit.

2.1.4. RALU

Most calculations performed by the 8096 take place in the RALU. The RALU, shown in Figure 2-2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16+ sign extension) wide. Some of the registers have the ability to perform simple operations to off-load the ALU.

A separate incrementer is used for the PC; however, jumps must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" register is used only when double-word quantities are being shifted, the "Upper Word" register is used

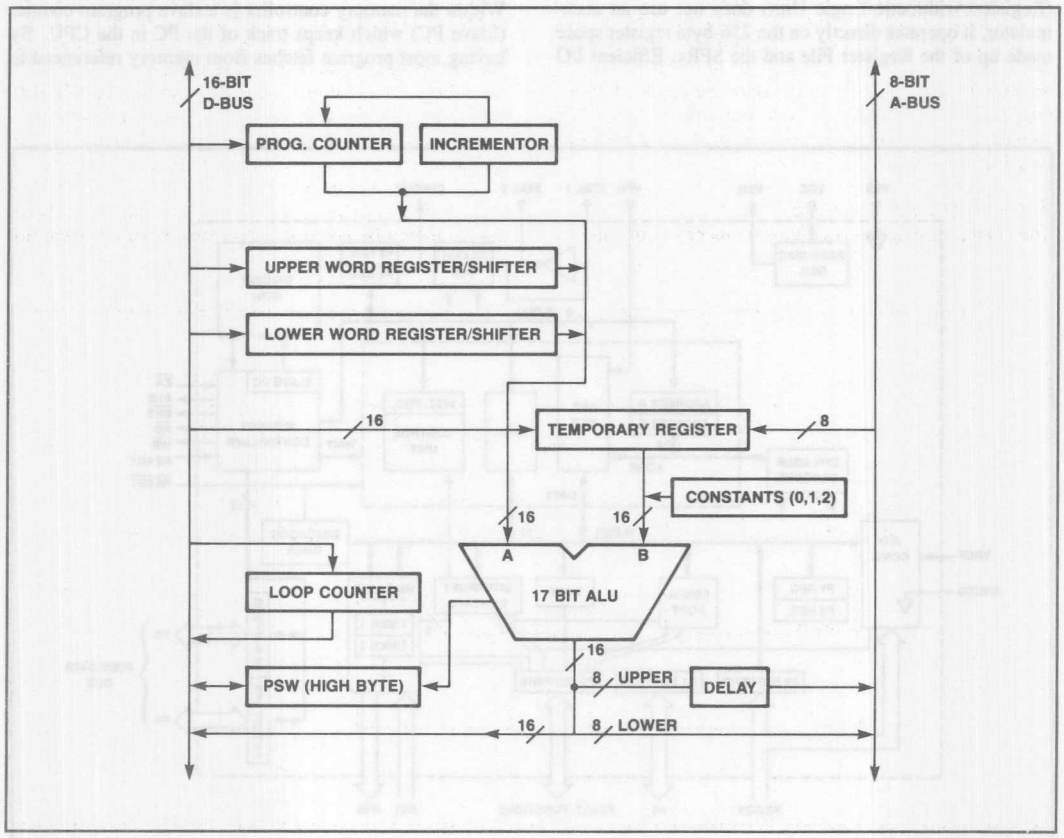


Figure 2-2. RALU Block Diagram

whenever a shift is performed or as a temporary register for many instructions. Repetitive shifts are counted by the 5-bit "Loop Counter".

A temporary register is used to store the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

The DELAY shown in Figure 2-2 is used to convert the 16-bit bus into an 8-bit bus. This is required as all addresses and instructions are carried on the 8-bit A bus. Several constants, such as 0, 1 and 2 are stored in the RALU for use in speeding up certain calculations. These come in handy when the RALU needs to make a 2's complement number or perform an increment or decrement instruction.

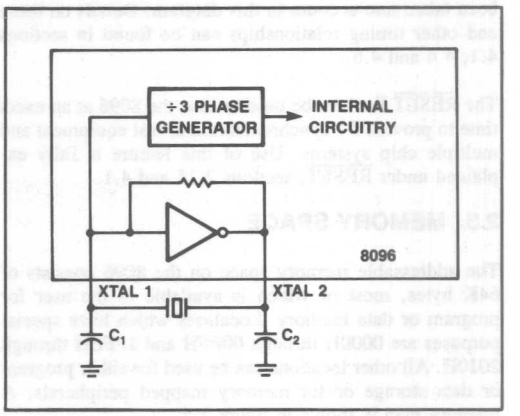


Figure 2-3. Block Diagram of Oscillator

2.2. BASIC TIMING

The 8096 requires an input clock frequency of between 6.0 MHz and 12 MHz to function. This frequency can be applied directly to XTAL1. Alternatively, since XTAL1 and XTAL2 are inputs and outputs of an inverter, it is also possible to use a crystal to generate the clock. A block diagram of the oscillator section is shown in Figure 2-3. Details of the circuit and suggestions for its use can be found in section 4.1.

2.2.1. Internal Timings

The crystal or external oscillator frequency is divided by 3 to generate the three internal timing phases as shown in Figure 2-4. Each of the internal phases repeat every 3 oscillator periods: 3 oscillator periods are referred to as one "state time", the basic time measurement for 8096 operations. Most internal operations are synchronized to either Phase A, B or C, each of which have a 33% duty cycle. Phase A is represented externally by CLKOUT, a signal available on the 68-pin part. Phases B and C are not available externally. The relationships of XTAL1, CLKOUT, and Phases A, B, and C are shown in Figure 2-4. It should be noted that propagation delays have not

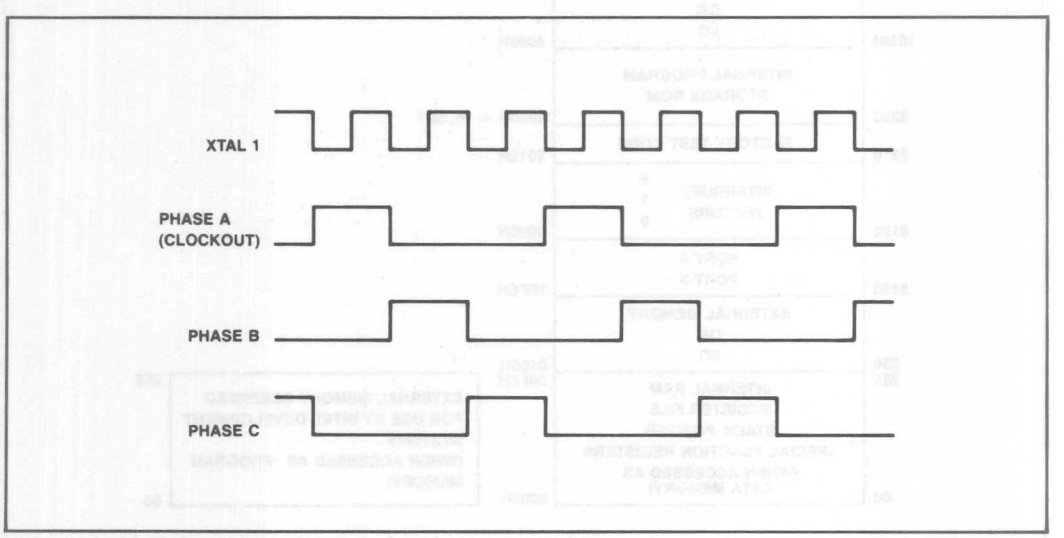


Figure 2-4. Internal Timings Relative to XTAL 1

been taken into account in this diagram. Details on these and other timing relationships can be found in sections 4.1, 4.4 and 4.6.

The $\overline{\text{RESET}}$ line can be used to start the 8096 at an exact time to provide for synchronization of test equipment and multiple chip systems. Use of this feature is fully explained under $\overline{\text{RESET}}$, sections 2.15 and 4.1.

2.3. MEMORY SPACE

The addressable memory space on the 8096 consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2010H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in figure 2-5.

2.3.1. Register File

Locations 00H through 0FFH contain the Register File and SFRs. Complete information on this section of memory space can be found in section "RAM SPACE". No code can be executed from this internal RAM section. If an attempt to execute instructions from locations 000H through 0FFH is made, the instructions will be fetched from *external* memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location 0000H, therefore, the NMI instruction is also reserved for Intel development tools.

2.3.2. Reserved Memory Spaces

Locations 1FFEH and 1FFFH are reserved for Ports 3 and 4 respectively. This is to allow easy reconstruction of these ports if external memory is used in the system. An example of reconstructing the I/O ports is given in section 4.6.7. If ports 3 and 4 are not going to be reconstructed then these locations can be treated as any other external memory location.

The 9 interrupt vectors are stored in locations 2000H through 2011H. The 9th vector is used by Intel development systems, as explained in section 2.5. Internal locations 2012H through 2077H are reserved for Intel's factory test code. These locations may be used in external memory.

Resetting the 8096 causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in section 2.15.

2.3.3. Internal ROM

When a ROM part is ordered, the internal memory locations 2080H through 3FFFH are user specified as are the interrupt vectors in locations 2000H through 2011H.

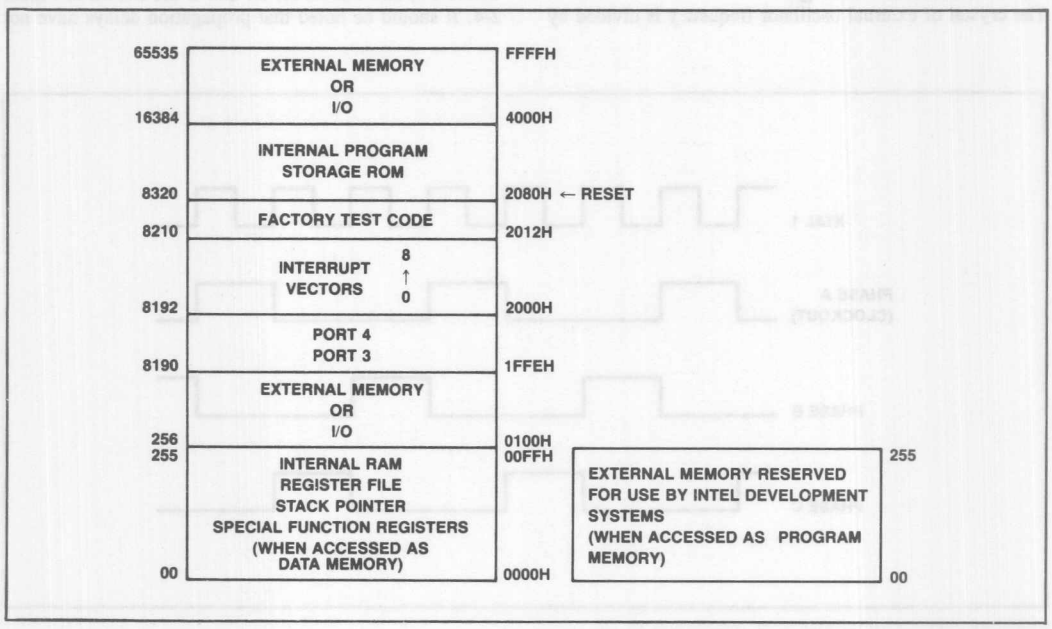


Figure 2-5. Memory Map

Instruction and data fetches from the internal ROM occur only if the part has a ROM, $\overline{EA}$ is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory.

2.3.4. Memory Controller

The RALU talks to the memory (except for the locations in the register file and SFR space) through the memory controller which is connected to the RALU by the A-bus and several control lines. Since the A-bus is eight bits wide, the memory controller uses a Slave Program Counter to avoid having to always get the instruction location from the RALU. This slave PC is incremented after each fetch. When a jump or call occurs, the slave PC must be loaded from the A-bus before instruction fetches can continue.

In addition to holding a slave PC, the memory controller contains a 3 byte queue to help speed execution. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Tables 3-3 and 3-4 show the normal execution times with no wait states added. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This is reflected in the jump taken/not-taken times shown in Table 3-4.

2.3.5. System Bus

External memory is addressed through lines AD0 through AD15 which form a 16-bit multiplexed (address/data) data bus. These lines share pins with I/O ports 3 and 4. The falling edge of the Address Latch Enable (ALE) line is used to provide a clock to a transparent latch (74LS373) in order to demultiplex the bus. A typical circuit and the required timings are shown in section 4.6. Since the 8096's external memory can be addressed as either bytes or words, the decoding is controlled with two lines, Bus High Enable ($\overline{BHE}$) and Addresss/Data Line 0 (AD0). The $\overline{BHE}$ line must be transparently latched, just as the addresses are.

To avoid confusion during the explanation of the memory system it is reasonable to give names to the demultiplexed address/data signals. The address signals will be called MA0 through MA15 (Memory Address), and the data signals will be called MD0 through MD15 (Memory Data).

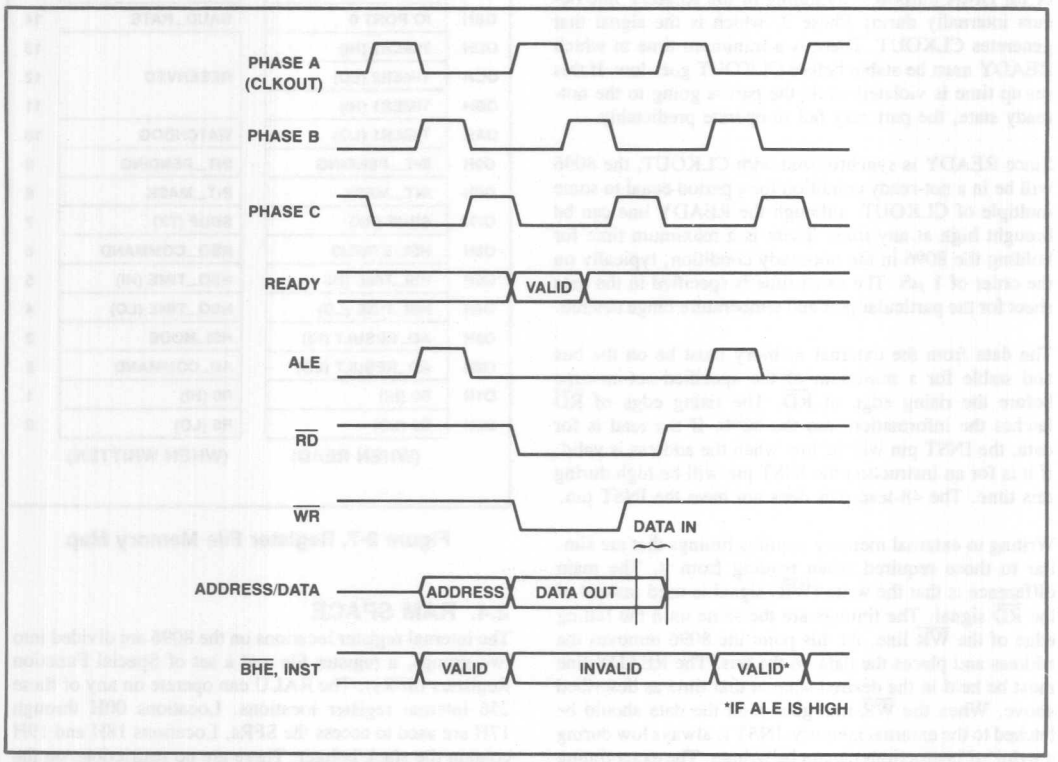


Figure 2-6. External Memory Timings

When $\overline{BHE}$ is active (low), the memory connected to the high byte of the data bus should be selected. When MA0 is low the memory connected to the low byte of the data bus should be selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only (MA0=0, $\overline{BHE}$=1), to the high (odd) byte only (MA0=1, $\overline{BHE}$=0), or to both bytes (MA0=0, $\overline{BHE}$=0). When a memory block is being used only for reads, $\overline{BHE}$ and MA0 need not be decoded.

Figure 2-6 shows the idealized waveforms related to the following description of external memory manipulations. For exact timing specifications please refer to the latest data sheet. When an external memory fetch begins, the address latch enable (ALE) line rises, the address is put on AD0-AD15 and $\overline{BHE}$ is set to the required state. ALE then falls, the address is taken off the pins, and the $\overline{RD}$ (Read) signal goes low. The READY line can be pulled low to hold the processor in this condition for a few extra state times.

2.3.6. Ready

The READY line can be used to hold the processor in the above condition in order to allow access to slow memories or for DMA purposes. Sampling of the READY line occurs internally during Phase 2, which is the signal that generates CLKOUT. There is a minimum time in which READY must be stable before CLKOUT goes low. If this set-up time is violated while the part is going to the not-ready state, the part may fail to operate predictably.

Since READY is synchronized with CLKOUT, the 8096 will be in a not-ready condition for a period equal to some multiple of CLKOUT, although the READY line can be brought high at any time. There is a maximum time for holding the 8096 in the not-ready condition, typically on the order of 1 μS. The exact time is specified in the data sheet for the particular part and temperature range desired.

The data from the external memory must be on the bus and stable for a minimum of the specified set-up time before the rising edge of $\overline{RD}$. The rising edge of $\overline{RD}$ latches the information into the 8096. If the read is for data, the INST pin will be low when the address is valid, if it is for an instruction the INST pin will be high during this time. The 48-lead part does not have the INST pin.

Writing to external memory requires timings that are similar to those required when reading from it. The main difference is that the write ($\overline{WR}$) signal is used instead of the $\overline{RD}$ signal. The timings are the same until the falling edge of the $\overline{WR}$ line. At this point the 8096 removes the address and places the data on the bus. The READY line must be held in the desired state at that time as described above. When the $\overline{WR}$ line goes high the data should be latched to the external memory. INST is always low during a write, as instructions cannot be written. The exact timing specifications for memory accesses can be found in the data sheet.

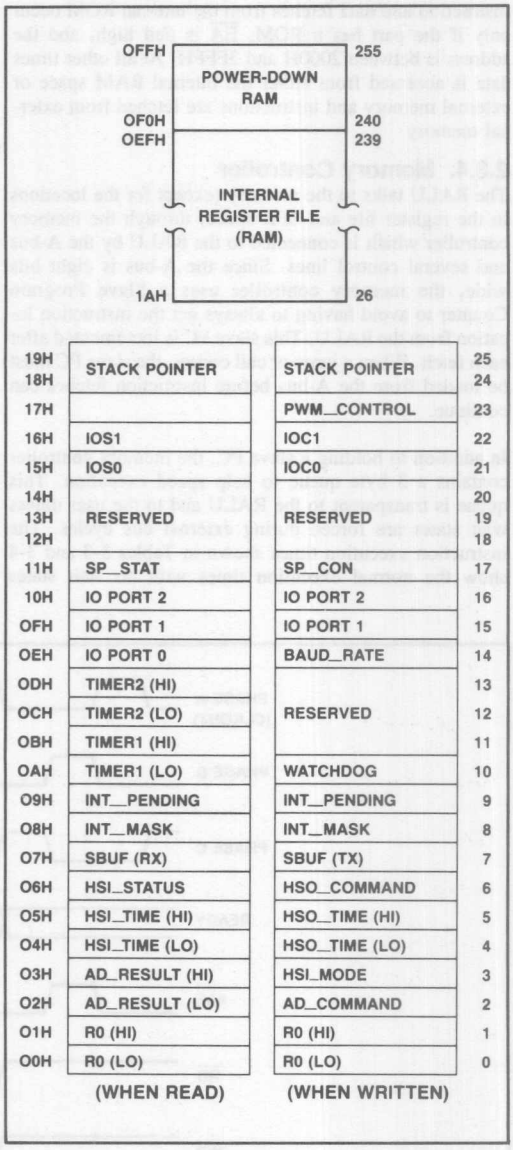


Figure 2-7. Register File Memory Map

2.4. RAM SPACE

The internal register locations on the 8096 are divided into two groups, a register file and a set of Special Function Registers (SFRs). The RALU can operate on any of these 256 internal register locations. Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. There are no restrictions on the use of the remaining 230 locations except that code cannot be executed from them.

2.4.1. Special Function Registers

All of the I/O on the 8096 is controlled through the SFRs. Many of these registers serve two functions; one if they are read from, the other if they are written to. Figure 2-7 shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown in Figure 2-8, with complete descriptions reserved for later chapters.

Register	Description	Chapter
R0	Zero Register — Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.	3.2.7
AD_RESULT	A/D Result Hi/Low — Low and high order Results of the A/D converter	2.9.3
AD_COMMAND	A/D Command Register — Controls the A/D	2.9.2
HSI_MODE	HSI Mode Register — Sets the mode of the High Speed Input unit.	2.7.1
HSI_TIME	HSI Time Hi/Lo — Contains the time at which the High Speed Input unit was triggered.	2.7.4
HSO_TIME	HSO Time Hi/Lo — Sets the time for the High Speed Output to execute the command in the Command Register.	2.8.3
HSO_COMMAND	HSO Command Register — Determines what will happen at the time loaded into the HSO Time registers.	2.8.2
HSI_STATUS	HSI Status Registers — Indicates which HSI pins were detected at the time in the HSI Time registers.	2.7.4
SBUF (RX)	Receive buffer for the serial port, holds contents to be outputed.	2.11
SBUF (TX)	Transmit buffer for the serial port, holds the byte just received by the serial port.	2.11
INT_MASK	Interrupt Mask Register — Enables or disables the individual interrupts.	2.5.2 3.6.2
INT_PENDING	Interrupt Pending Register — Indicates when an interrupt signal has occurred on one of the sources.	2.5.2 3.6.2
WATCHDOG	Watchdog Timer Register — Written to periodically to hold off automatic reset every 64K state times.	2.14
TIMER1	Timer 1 Hi/Lo — Timer 1 high and low bytes.	2.6.1 2.7-8
TIMER2	Timer 2 Hi/Lo — Timer 2 high and low bytes.	2.6.2 2.7-8
IOPORT0	Port 0 Register — Levels on pins of port 0.	2.12.1
BAUD_RATE	Register which contains the baud rate, this register is loaded sequentially.	2.11.4
IOPORT1	Port 1 Register — Used to read or write to Port 1.	2.12.2
IOPORT2	Port 2 Register — Used to read or write to Port 2.	2.12.3
SP_STAT	Serial Port Status — Indicates the status of the serial port.	2.11.3
SP_CON	Serial port control — Used to set the mode of the serial port.	2.11.1
IOS0	I/O Status Register 0 — Contains information on the HSO status.	2.13.4
IOS1	I/O Status Register 1 — Contains information on the status of the timers and of the HSI.	2.13.5 3.7.2
IOC0	I/O Control Register 0 — Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.	2.13.2
IOC1	I/O Control Register 1 — Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.	2.13.3
PWM_CONTROL	Pulse Width Modulation Control Register — Sets the duration of the PWM pulse.	2.10 4.3.2

Figure 2-8. SFR Summary

Within the SFR space are several registers labeled as "RESERVED". These registers are reserved for future expansion or test purposes. Reads or writes of these registers may produce unexpected results. For example, writing to location 000CH will set both timers to 0FFFXH, this feature is for use in testing the part and should not be used in programs.

2.4.2. Power Down

The upper 16 RAM locations (0F0H through 0FFH) receive their power from both the VCC pin and the VPD pin. If it is desired to keep the memory in these locations alive during a power down situation, one need only keep voltage on the VPD pin. The current required to keep the RAM alive is approximately 1 milliamp (refer to the data sheet for the exact specification).

To place the 8096 into a power down mode, the $\overline{\text{RESET}}$ pin is pulled low. Two state times later the part will be in reset. This is necessary to prevent the part from writing into RAM as the power goes down. The power may now be removed from the VCC pin, the VPD pin must remain within specifications. The 8096 can remain in this state for any amount of time and the 16 RAM bytes will retain their values.

To bring the 8096 out of power down, $\overline{\text{RESET}}$ is held low while VCC is applied. Two state times after the oscillator and the back bias generator have stabilized (~1 millisecond), the $\overline{\text{RESET}}$ pin can be pulled high. The 8096 will begin to execute code at location 02080H 10 state times after $\overline{\text{RESET}}$ is pulled high. Figure 2-9 shows a timing diagram of the power down sequence. To ensure that the 2 state time minimum reset time (synchronous with CLKOUT) is met, it is recommended that 10 XTAL1 cycles be used. Suggestions for actual hardware connections are given in section 4.1. Reset is discussed in section 2.15.

2.5. INTERRUPT STRUCTURE

2.5.1. Interrupt Sources

Eight interrupt sources are available on the 8096. When enabled, an interrupt occurring on any of these sources will force a call to the location stored in the vector location for that source. The interrupt sources and their respective vector locations are listed in Figure 2-10. In addition to the 8 standard interrupts, there is a TRAP instruction which acts as a software generated interrupt. This instruction is not currently supported by the MCS-96 Assembler and is reserved for use by Intel development systems. Many of the interrupt sources can be activated by several methods, Figure 2-11 shows all of the possible sources for interrupts.

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-10. Interrupt Vector Locations

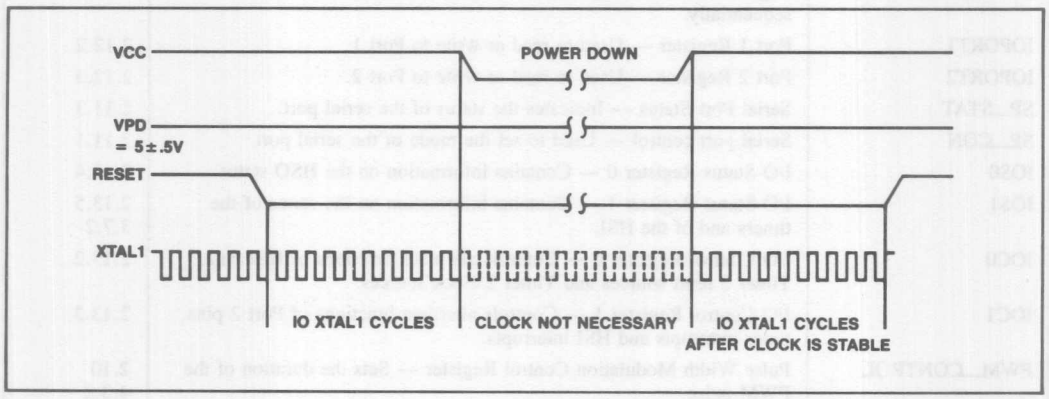


Figure 2-9. Power Down Timing

2.5.2. Interrupt Control

A block diagram of the interrupt system is shown in Figure 2-12. Each of the interrupt sources is tested for a 0 to 1 transition. If this transition occurs, the corresponding bit in the Interrupt Pending Register, located at 0009H, is set. Since this register can be written to, it is possible to generate software interrupts by setting bits within the register, or remove pending interrupts by clearing the bits in this register. The pending register can be set even if the interrupt is disabled.

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 4 state time "partial" interrupt cycle will occur. This is because the 8096 will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra NOP. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 8096 holds off acknowledging interrupts during these "read/modify/write" instructions.

Enabling and disabling of individual interrupts is done through the Interrupt Mask Register, located at 0008H. If the bit in the mask register is a 1 then the interrupt is enabled, otherwise it is disabled. Even if an interrupt is masked it may still become pending. It may, therefore, be desirable to clear the pending bit before unmasking an interrupt.

The Interrupt Mask Register is also the low byte of the PSW. All of the interrupts may be enabled and disabled simultaneously by using the "EI" (Enable Interrupt) and "DI" (Disable Interrupt) instructions. EI and DI set and clear PSW.9, the interrupt enable bit, they do not effect the contents of the mask register.

2.5.3. Interrupt Priority Programming

The priority encoder looks at all of the interrupts which are *both pending and enabled,* and selects the one with the highest priority. The priorities are shown in Figure 2-10 (7 is highest, 0 is lowest.) The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

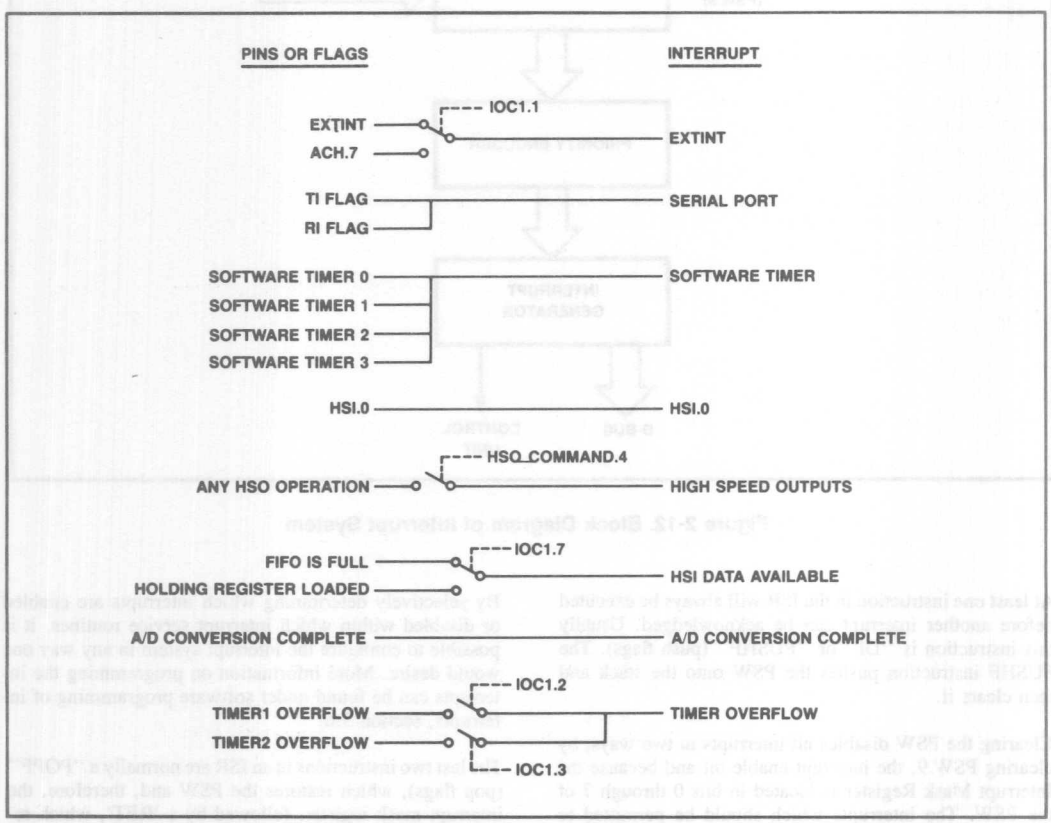


Figure 2-11. All Possible Interrupt Sources

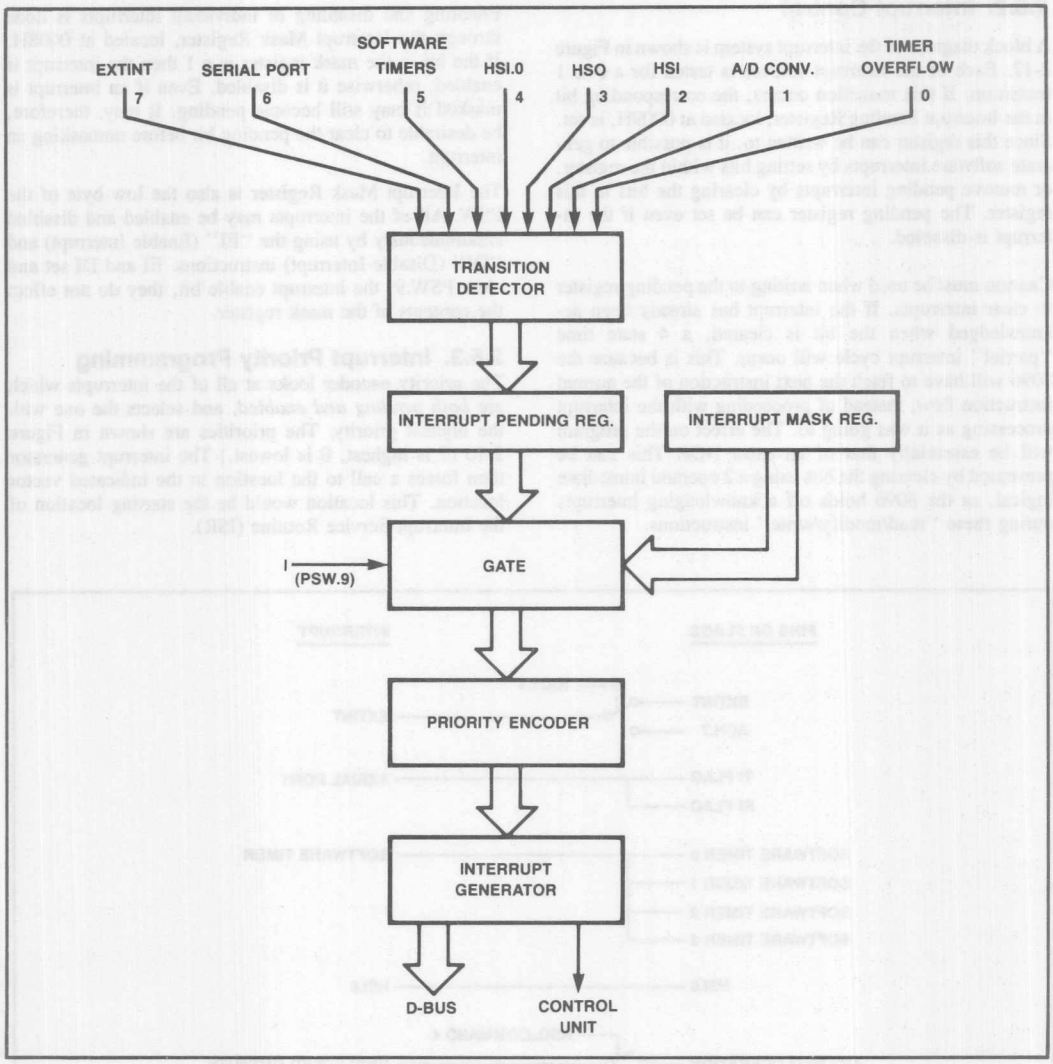


Figure 2-12. Block Diagram of Interrupt System

At least one instruction in the ISR will always be executed before another interrupt can be acknowledged. Usually this instruction is "DI" or "PUSHF" (push flags). The PUSHF instruction pushes the PSW onto the stack and then clears it.

Clearing the PSW disables all interrupts in two ways; by clearing PSW.9, the interrupt enable bit and because the Interrupt Mask Register is located in bits 0 through 7 of the PSW. The interrupts which should be permitted to interrupt this ISR can then be set in the mask register and an "EI" instruction executed.

By selectively determining which interrupts are enabled or disabled within which interrupt service routines, it is possible to configure the interrupt system in any way one would desire. More information on programming the interrupts can be found under software programming of interrupts, section 3.6.

The last two instructions in an ISR are normally a "POPF" (pop flags), which restores the PSW and, therefore, the interrupt mask register, followed by a 'RET', which restores the Program Counter. Execution will then continue from the point at which the call was forced.

2.5.4. Interrupt Timing

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt will not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

EI, DI	— Enable and Disable Interrupts
POPF, PUSHF	— Pop and Push Flags
SIGND	— Prefix to perform signed multiply and divide
TRAP	— Software interrupt

When an interrupt is acknowledged, a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 21 state times. If the stack is in external RAM an additional 3 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 8096 begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 43 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (21 to 24 state times). Therefore, the maximum response time is 71 (43+4+24) state times. This does not include the 12 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled.

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The "DI", "PUSHF", "POPF" and "TRAP" instructions will also cause the same situation. Typically the PUSHF, POPF and TRAP instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

2.6. TIMERS

Two 16-bit timers are available for use on the 8096. The first is designated "Timer 1", the second, "Timer 2". Timer 1 is used to synchronize events to real time, while Timer 2 can be clocked externally and synchronizes events to external occurences.

2.6.1. Timer 1

Timer 1 is clocked once every eight state times and can be cleared only by executing a reset. The only other way to change its value is by writing to 000CH but this is a test mode which sets both timers to 0FFFXH and should not be used in programs.

2.6.2. Timer 2

Timer 2 can be incremented by transitions (one count each transition, rising *and* falling) on either T2CLK or HSI.1. The multiple functionality of the timer is determined by the state of I/O Control Register 0, bit 7 (IOC0.7). To ensure that all CAM entries are checked each count of Timer 2, the maximum transition speed is limited to once per eight state times. Timer 2 can be cleared by: executing a reset, by setting IOC0.1, by triggering HSO channel 0EH, or by pulling T2RST or HSI.0 high. The HSO and CAM are described in section 2.8. IOC0.3 and IOC0.5 control the resetting of Timer 2. Figure 2-13 shows the different ways of manipulating Timer 2.

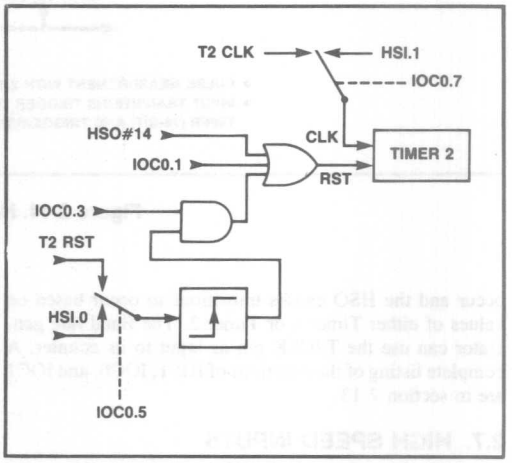


Figure 2-13. Timer 2 Clock and Reset Options

2.6.3. Timer Interrupts

Both Timer 1 and Timer 2 can be used to trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). The interrupts are controlled by IOC1.2 and IOC1.3 respectively. The flags are set in IOS1.6 and IOS1.7, respectively.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears the whole byte, including the software timer flags. It is, therefore, recommended to write the byte to a temporary register before testing bits. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.

2.6.4. Timer Related Sections

The High Speed I/O unit is coupled to the timers in that the HSI records the value on Timer 1 when transitions

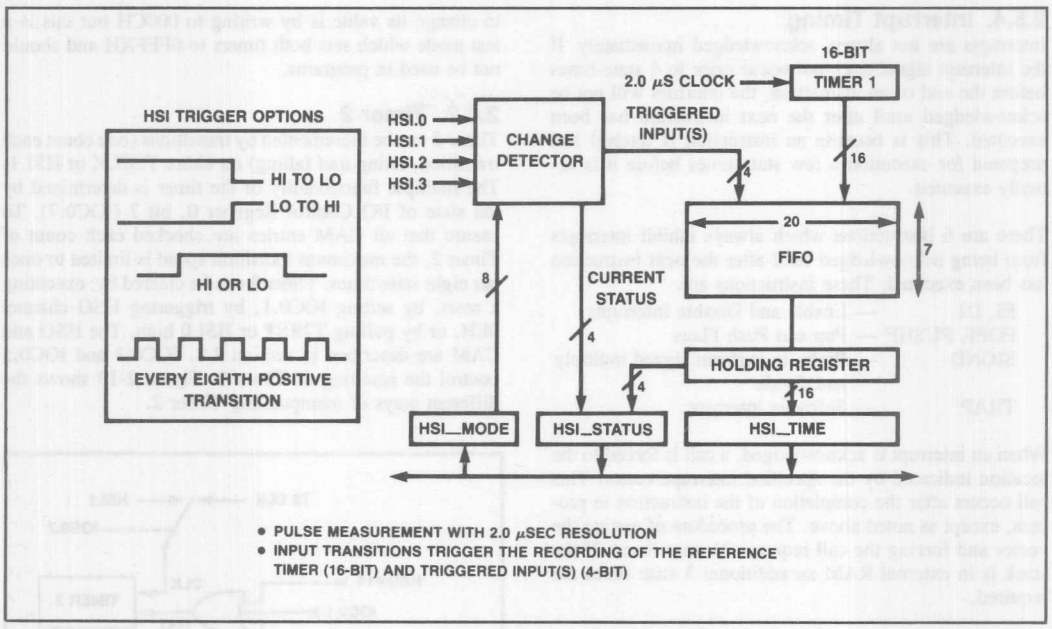


Figure 2-14. High Speed Input Unit

occur and the HSO causes transitions to occur based on values of either Timer 1 or Timer 2. The Baud rate generator can use the T2CLK pin as input to its counter. A complete listing of the functions of IOS1, IOC0, and IOC1 are in section 2.13.

2.7. HIGH SPEED INPUTS

The High Speed Input Unit (HSI), can be used to record the time at which an event occurs with respect to Timer 1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 share pins with HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) are used to determine the functions of these pins. A block diagram of the HSI unit is shown in Figure 2-14.

2.7.1. HSI Modes

There are 4 possible modes of operation for each of the HSI. The HSI mode register is used to control which pins will look for what type of events. The 8-bit register is set up as shown in Figure 2-15.

High and low levels each need to be held for at least 1 state time to ensure proper operation. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time.

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 2-16 shows

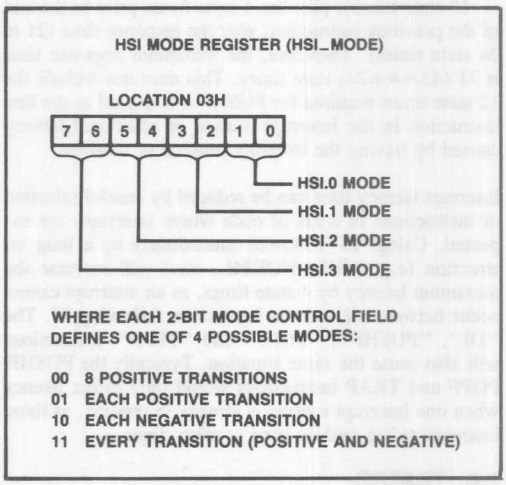


Figure 2-15. HSI Mode Register Diagram

the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO.

2.7.2. HSI FIFO

When an HSI event occurs, a 7x20 FIFO stores the 16 bits of Timer 1 and the 4 bits indicating the state of the

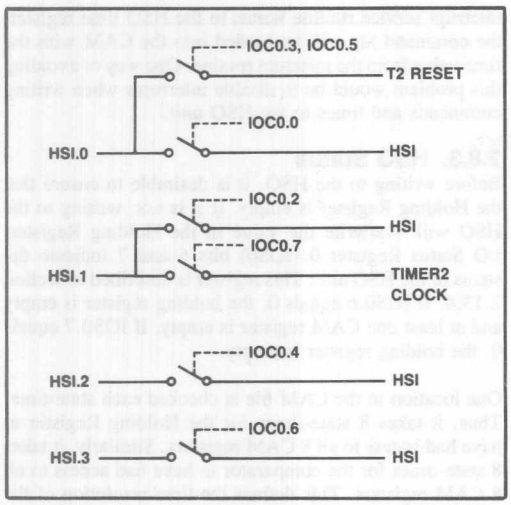


Figure 2-16. IOC0 Control of HSI Pin Functions

4 HSI lines at the recorded timer value. It can take up to 8 state times for this information to reach the holding register. When the FIFO is full, one additional event can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full any additional events will not be recorded.

2.7.3. HSI Interrupts

Interrupts can be generated from the HSI unit in one of two ways, determined by IOC1.7. If the bit is a 0, then an interrupt will be generated every time a value is loaded into the holding register. If it is a 1, an interrupt will only be generated when the FIFO, (independent of the holding register), has six or seven entries in it. Since all interrupts are rising edge triggered, if IOC1.7 = 1, the processor will not be re-interrupted until the FIFO first contains 5 or less records, then contains six or more.

2.7.4. HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. If bit 6 is a 1, the FIFO contains at least seven entries. If bit 7 is a 1, the FIFO contains at least 1 entry. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in IOS1, as this action clears the entire byte, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value.

Reading the HSI is done in two steps. First, the HSI Status register is read to obtain the current state of the HSI pins and their state at the recorded time. Second, the HSI Time register is read. Reading the Time register unloads one word of the FIFO, so if the Time register is read before the Status register, the information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

It should be noted that many of the Status register conditions are changed by a reset, see section 2.15.2. A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13.

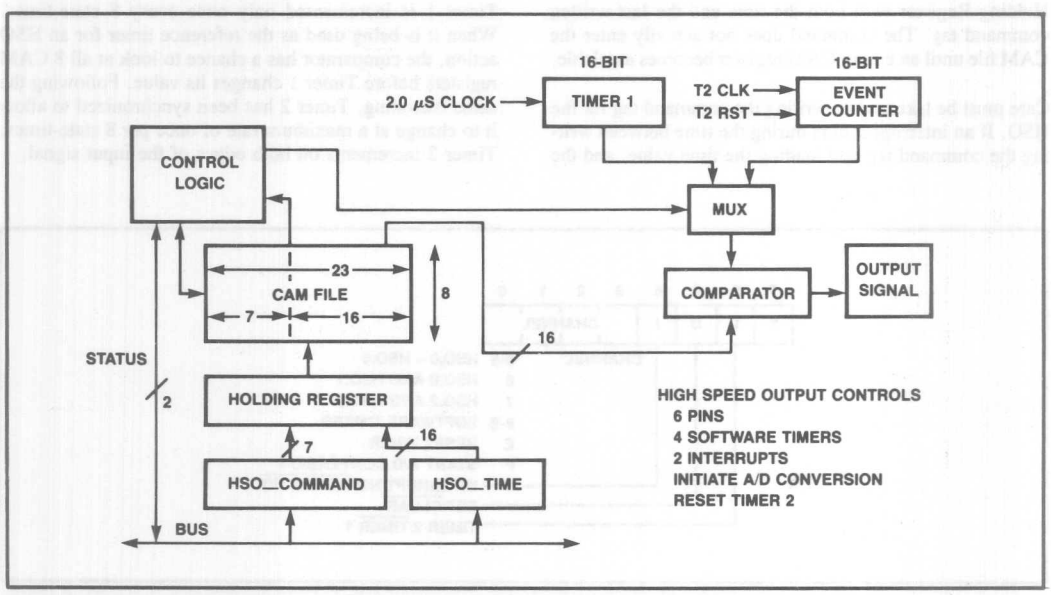


Figure 2-17. High Speed Output Unit

2.8. HIGH SPEED OUTPUTS

The High Speed Output unit (HSO) is used to trigger events at specific times with minimal CPU overhead. These events include: starting an A to D conversion, resetting Timer 2, setting 4 software flags, generating interrupts, and switching up to 6 output lines. Up to 8 events can be pending at any one time.

2.8.1. HSIO Shared Pins

Two of the 6 output lines (HSO.0 through HSO.5) are shared with the High Speed Input (HSI) lines. HSO.4 and HSO.5 are shared with HSI.2 and HSI.3, respectively. Bits 4 and 6 of the I/O Control Register 1 (IOC1) are used to enable HSO.4 and HSO.5 as outputs.

2.8.2. HSIO CAM

A block diagram of the HSO unit is shown in Figure 2-17. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with a time value every state time. Therefore, it takes 8 state times to compare all CAM registers with a timer.

Each CAM register is 23 bits wide. Sixteen bits specify the time at which the action is to be carried out and 7 bits specify both the nature of the action and whether Timer 1 or Timer 2 is the reference. The format of the command to the HSO unit shown in Figure 2-18.

To enter a command into the CAM file, write the 7-bit "Command Tag" into location 0006H followed by the time at which the action is to be carried out into word address 0004H. Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag will be loaded into the CAM with the time value from the interrupt routine. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit.

2.8.3. HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOS0) bits 6 and 7 indicate the status of the HSO unit. This register is described in section 2.13.4. If IOS0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty.

One location in the CAM file is checked each state-time. Thus, it takes 8 state-times for the Holding Register to have had access to all 8 CAM registers. Similarly, it takes 8 state-times for the comparator to have had access to all 8 CAM registers. This defines the time-resolution of the HSO unit to be 8 state-times (2.0 μsec, if the oscillator frequency is 12 MHz). Note that the comparator does not look at the holding register, so instructions in the holding register do not execute.

2.8.4. Clearing The HSO

All 8 CAM locations of the HSO are compared before any action is taken. This allows a pending event to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer matches the written value or the chip is reset.

Timer 1 is incremented only once every 8 state-times. When it is being used as the reference timer for an HSO action, the comparator has a chance to look at all 8 CAM registers before Timer 1 changes its value. Following the same reasoning, Timer 2 has been synchronized to allow it to change at a maximum rate of once per 8 state-times. Timer 2 increments on both edges of the input signal.

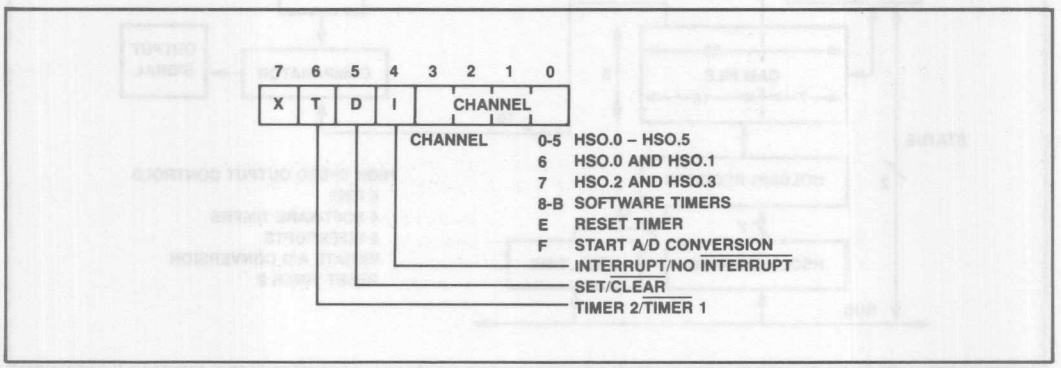


Figure 2-18. HSO Command Tag Format

If Timer 2 is being used as the reference timer, the user must ensure that it is not cleared before all references to it in the CAM have been recognized. If this occurs, the unrecognized references will remain in the CAM, blocking further entries to those CAM registers until the part is reset or the references are recognized.

2.8.5. Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preprogrammed time is reached, the HSO unit generates a Software Timer interrupt. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer 2 or starts an A to D conversion, it can also be programmed to generate a software timer interrupt but there is no flag to indicate that this has occurred.

Each read or test of any bit in IOS1 will clear the whole byte. Be certain to save the byte before testing it unless you are only concerned with 1 bit.

A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13. The Timers are described in section 2.6 and the HSI is described in section 2.7.

2.9. ANALOG INPUTS

The A to D converter on the 8096 provides a 10-bit result on one of 8 input channels. Conversion is done using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D converter is available on the 8097, 8397, 8095 and 8395 members of the MCS®-96 family.

2.9.1. A/D Accuracy

Each conversion requires 168 state-times (42μS at 12 MHz) independent of the accuracy desired or value of input voltage. The input voltage must be in the range of 0 to VREF, the analog reference and supply voltage. For proper operation, VREF (the reference voltage and analog power supply) must be held at VCC $\pm$ 0.3V with VREF = 5.0 $\pm$ 0.5V. The A/D result is calculated from the formula:

$$1023 \times (\text{input voltage-AVGND}) / (\text{VREF-AVGND})$$

It can be seen from this formula that changes in VREF or AVGND effect the output of the converter. This can be advantageous if a ratiometric sensor is used since these sensors have an output that can be measured as a proportion of VREF.

If high absolute accuracy is needed it may be desirable to use a separate power supply, or power traces, to operate the A/D converter. There is no sample and hold circuit internal to the chip, so the input voltage must be held constant for the entire 168 state times. Examples of connecting the A/D converter to various devices are given in section 4.3.

2.9.2. A/D Commands

Analog signals can be sampled by any one of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. ACH7 can also be used as an external interrupt if IOC1.1 is set (see section 2.5). The A/D Command Register, at location 02H, selects which channel is to be converted and whether the conversion should start immediately or when the HSO (Channel #14) triggers it. A to D commands are formatted as shown in Figure 2-19.

The command register is double buffered so it is possible

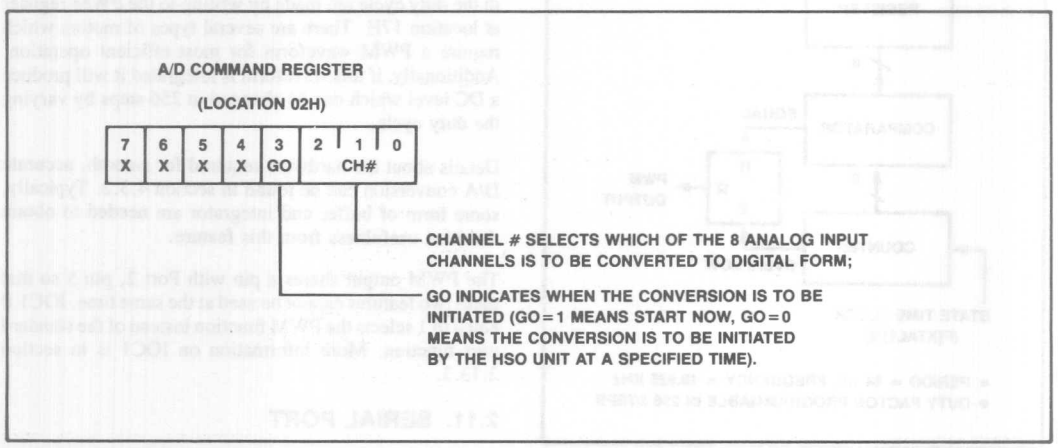


Figure 2-19. A/D Command Register

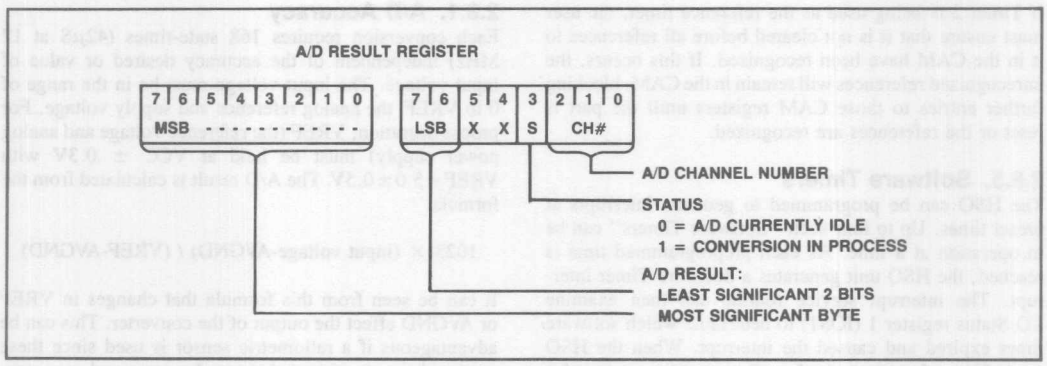


Figure 2-20. A/D Result Register

to write a command to start a conversion triggered by the HSO while one is still in progress. Care must be taken when this is done since if a new conversion is started while one is already in progress, the conversion in progress is cancelled and the new one is started. When a conversion is started, the result register is cleared. For this reason the result register must be read before a new conversion is started or data will be lost.

2.9.3. A/D Results

Results of the analog conversions are read from the A/D Result Register at locations 02H and 03H. Although these addresses are on a word boundary, they must be read as individual bytes. Information in the A/D Result register is formatted as shown in Figure 2-20. Note that the status bit may not be set until 8 state times after the go command. Information on using the HSO is in section 2.8.

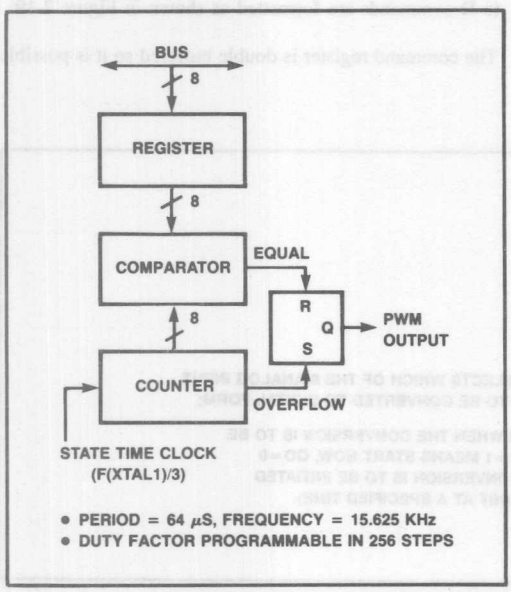


Figure 2-21. Pulse Width Modulated (D/A) Output

2.10. PULSE WIDTH MODULATION OUTPUT (D/A)

Digital to analog conversion can be done with the pulse width modulation output; a block diagram of the circuit is shown in Figure 2-21. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 2-22. Note that when the PWM register equals 00, the output is always low.

The output waveform is a variable duty cycle pulse which repeats every 256 state times (64 μS at 12MHz). Changes in the duty cycle are made by writing to the PWM register at location 17H. There are several types of motors which require a PWM waveform for most efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle.

Details about the hardware required for smooth, accurate D/A conversion can be found in section 4.3.2. Typically, some form of buffer and integrator are needed to obtain the most usefulness from this feature.

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOC1.0 equal to 1 selects the PWM function instead of the standard port function. More information on IOC1 is in section 2.13.3.

2.11. SERIAL PORT

The serial port is compatible with the MCS-51 serial port. It is full duplex, meaning it can transmit and receive si-

multaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. The serial port registers are both accessed at location 07H. A write to this location accesses the transmit register, and a read accesses a physically separate receive register.

The serial port can operate in 4 modes (explained below). Selection of these modes is done through the Serial Port Control register at location 11H. Use of this register will be described after defining the possible modes of operation. In addition to setting the proper mode, it is necessary to enable the TXD output by setting IOC1.5 high before the serial port is used.

2.11.1. Serial Port Modes

MODE 0

Mode 0 is a shift register mode. The 8096 outputs a train of 8 shift pulses to an external shift register to clock 8 bits of data into or out of the register from or to the 8096. Serial data enters and exits the 8096 through RXD. TXD outputs the shift clock. 8 bits are transmitted or received, LSB first. A timing diagram of this mode is shown in Figure 2-23. This mode is useful as an I/O expander in which application external shift registers can be used as additional parallel I/O ports. An example of using the port in this mode is given in section 4.5.

MODE 1

10-bit frames are transmitted through TXD, and received through RXD: a start bit (0), 8 data bits (LSB first), and a stop bit (1). This mode is the one commonly used for CRT terminals. The data frame for Mode 1 is shown in Figure 2-24.

MODE 2

11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1. On receive, the serial port interrupt is not activated unless the received 9th data bit is 1. This mode is commonly used along with mode 3 in a multiprocessor environment.

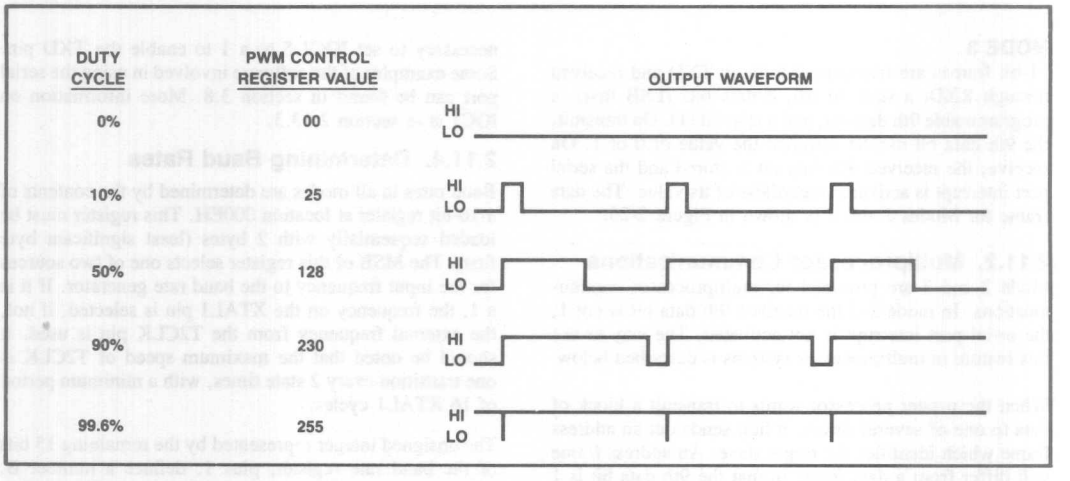


Figure 2-22. Typical PWM Outputs

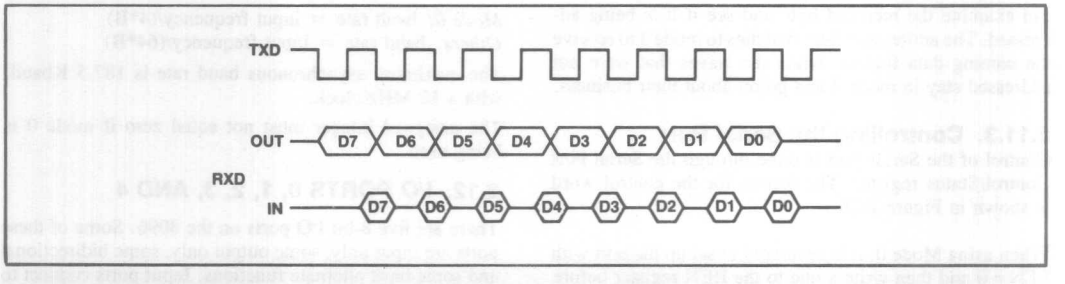


Figure 2-23. Serial Port Mode 0 Timing

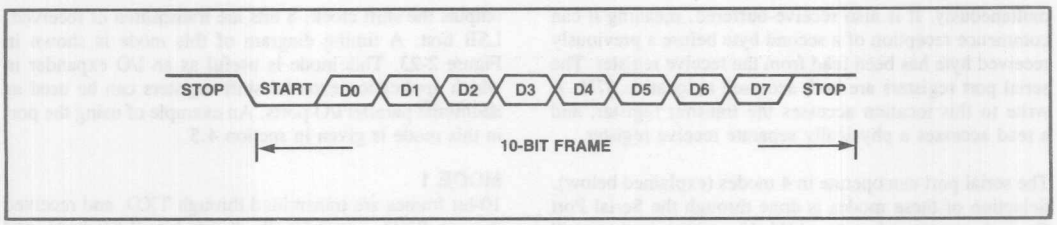


Figure 2-24. Serial Port Frame — Mode 1

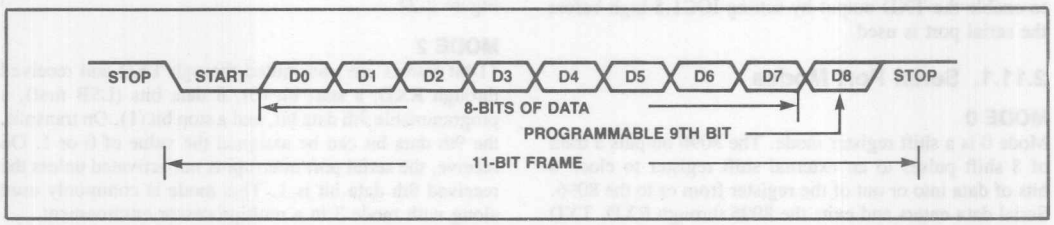


Figure 2-25. Serial Port Frame Modes 2 and 3

MODE 3

11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1. On receive, the received 9th data bit is stored and the serial port interrupt is activated regardless of its value. The data frame for Modes 2 and 3 is shown in Figure 2-25.

2.11.2. Multiprocessor Communications

Mode 2 and 3 are provided for multiprocessor communications. In mode 2 if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multiprocessor systems is described below.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. No slave in mode 2 will be interrupted by a data frame. An address frame, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to mode 3 to receive the coming data frames, while the slaves that were not addressed stay in mode 2 and go on about their business.

2.11.3. Controlling the Serial Port

Control of the Serial Port is done through the Serial Port Control/Status register. The format for the control word is shown in Figure 2-26.

When using Mode 0, it is necessary to set up the port with REN = 0 and then write a one to the REN register before the port will be to properly initialized. In any mode, it is necessary to set IOC1.5 to a 1 to enable the TXD pin. Some examples of the software involved in using the serial port can be found in section 3.8. More information on IOC1 is in section 2.13.3.

2.11.4. Determining Baud Rates

Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. This register must be loaded sequentially with 2 bytes (least significant byte first). The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum speed of T2CLK is one transition every 2 state times, with a minimum period of 16 XTAL1 cycles.

The unsigned integer represented by the remaining 15 bits of the baud rate register, plus 1, defines a number B, where B can thus take any value from 1 to 32768. This '1' is not added if the T2CLK is used as the time base. The baud rate for the 4 serial modes is then given by

Mode 0: baud rate = input frequency/(4*B)
Others: baud rate = input frequency/(64*B)

The maximum asynchronous baud rate is 187.5 Kbaud, with a 12 MHz clock.

The unsigned integer must not equal zero if mode 0 is being used.

2.12. I/O PORTS 0, 1, 2, 3, AND 4

There are five 8-bit I/O ports on the 8096. Some of these ports are input only, some output only, some bidirectional and some have alternate functions. Input ports connect to the internal bus through an input buffer. Output ports

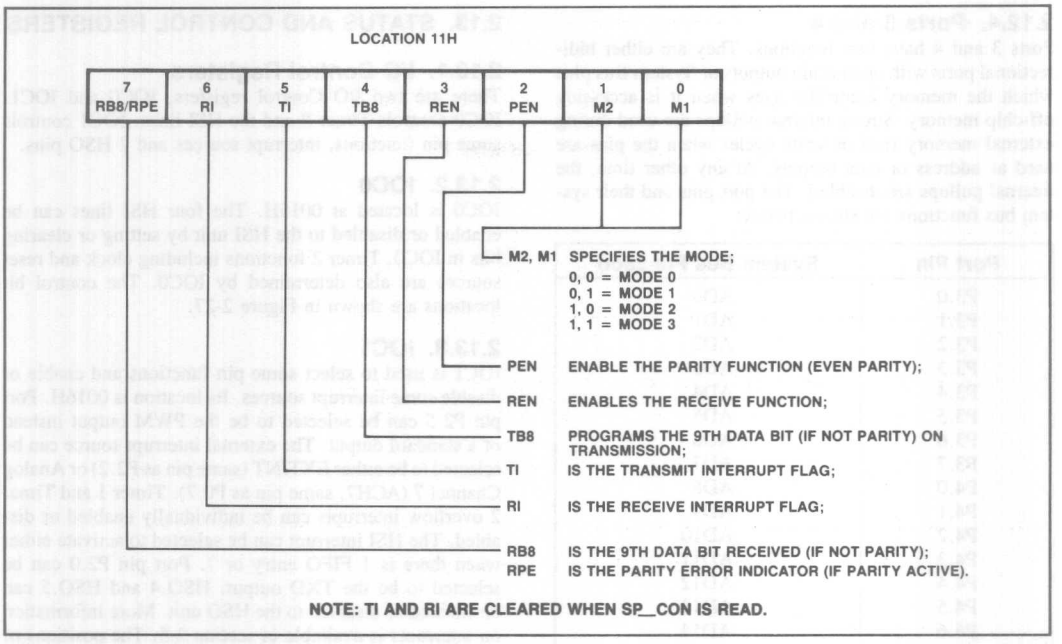


Figure 2-26. Serial Port Control/Status Register

connect through an output buffer to an internal register that holds the output bits. Bidirectional ports consist of an internal register, an output buffer, and an input buffer.

When an instruction accesses a bidirectional port as a source register, the question often arises as to whether the value that is brought into the CPU comes from the internal port register or from the port pins through the input buffer. In the 8096, the value always comes from the port pins, never from the internal register.

2.12.1. Port 0

Port 0 is an input only port which shares its pins with the analog inputs to the A/D Converter. One can read Port 0 digitally, and/or, by writing the appropriate control bits to the A/D Command Register, select one of the lines of this port to be the input to the A/D Converter. While a conversion is in process, the impedance of the selected line is lower than normal. See the data sheet for the specific values.

2.12.2. Port 1

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). If the internal pulldown is left off (by writing a 1 to the pin), the pin's logic level can be controlled by an external pulldown which can either be on (to input a 0) or off (to input a 1). From the user's point of view the main distinction is that a quasi-bidirectional input will source current while being externally held low and will pull itself high if left alone.

In parallel with the weak internal pullup, is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

2.12.3. Port 2

Port 2 is a multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

Port	Function	Alternate Function	Controlled by
P2.0	output	TXD (serial port transmit)	IOC1.5
P2.1	input	RXD (serial port receive)	N/A
P2.2	input	EXTINT (external interrupt)	IOC1.1
P2.3	input	T2CLK (Timer 2 input)	IOC0.7
P2.4	input	T2RST (Timer 2 reset)	IOC0.3
P2.5	output	PWM (pulse-width modulation)	IOC1.0
P2.6	quasi-bidirectional		
P2.7	quasi-bidirectional		

2.12.4. Ports 3 and 4

Ports 3 and 4 have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accessing off-chip memory. Strong internal pullups are used during external memory read or write cycles when the pins are used as address or data outputs. At any other time, the internal pullups are disabled. The port pins and their system bus functions are shown below:

Port Pin	System Bus Function
P3.0	AD0
P3.1	AD1
P3.2	AD2
P3.3	AD3
P3.4	AD4
P3.5	AD5
P3.6	AD6
P3.7	AD7
P4.0	AD8
P4.1	AD9
P4.2	AD10
P4.3	AD11
P4.4	AD12
P4.5	AD13
P4.6	AD14
P4.7	AD15

2.13. STATUS AND CONTROL REGISTERS

2.13.1. I/O Control Registers

There are two I/O Control registers, IOC0 and IOC1. IOC0 controls Timer 2 and the HSI lines. IOC1 controls some pin functions, interrupt sources and 2 HSO pins.

2.13.2. IOC0

IOC0 is located at 0015H. The four HSI lines can be enabled or disabled to the HSI unit by setting or clearing bits in IOC0. Timer 2 functions including clock and reset sources are also determined by IOC0. The control bit locations are shown in Figure 2-27.

2.13.3. IOC1

IOC1 is used to select some pin functions and enable or disable some interrupt sources. Its location is 0016H. Port pin P2.5 can be selected to be the PWM output instead of a standard output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit. More information on interrupts is available in section 2.5. The positions of the IOC1 control bits are shown in Figure 2-28.

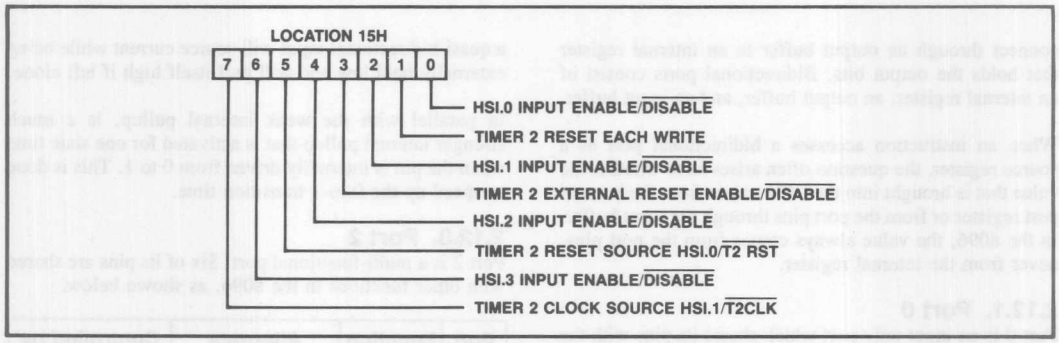


Figure 2-27. I/O Control Register 0 (IOC0)

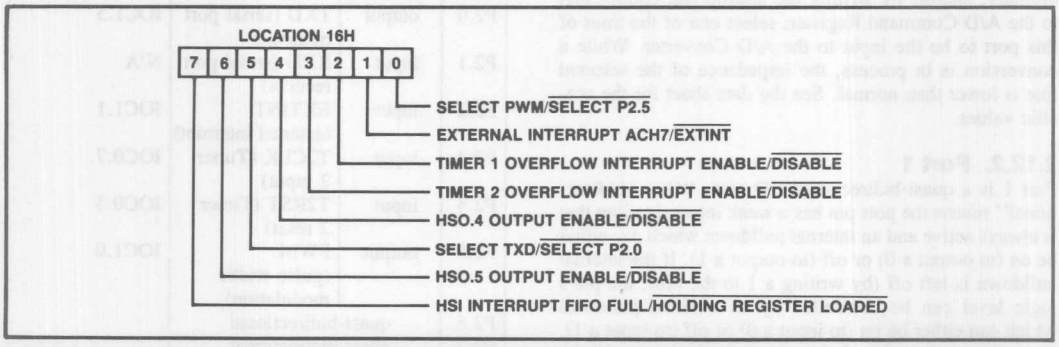


Figure 2-28. I/O Control Register 1 (IOC1)

2.13.4. I/O Status Register 0

There are two I/O Status registers, IOS0 and IOS1. IOS0, located at 0015H, holds the current status of the HSO lines and CAM. The status bits of IOS0 are shown in Figure 2-29.

2.13.5. I/O Status Register 1

IOS1 is located at 0016H. It contains status bits for the timers and the HSI. Every access of this register clears all of the timer overflow and timer expired bits. It is, therefore, important to first store the byte in a temporary location before attempting to test any bit unless only one bit will ever be of importance to the program. The status bits of IOS1 are shown in Figure 2-30.

2.14. WATCHDOG TIMER

This feature is provided as a means of graceful recovery from a software upset. If the software fails to reset the watchdog at least every 64K state times, a hardware reset will be initiated.

The software can be designed so that the watchdog times out if the program does not progress properly. The watchdog will also time-out if the software error was due to ESD (Electrostatic Discharge) or other hardware related problems. This prevents the controller from having a malfunction for longer than 16 mS if a 12 MHz oscillator is used.

The watchdog timer is a 16-bit counter which is incremented every state time. When it overflows it pulls down the $\overline{\text{RESET}}$ pin for at least two state times, resetting the 8096 and any other chips connected to the reset line. To prevent the timer from overflowing and resetting the system, it must be cleared periodically. Clearing the timer is done by writing a "01EH" followed by an "0E1H" to the WDT register at location 000AH.

Use of a large reset capacitor on the $\overline{\text{RESET}}$ pin will increase the length of time required for a watchdog initiated reset. This is because the capacitor will keep the $\overline{\text{RESET}}$ pin from responding immediately to the internal pull-ups and pull-downs. A large capacitor on the $\overline{\text{RESET}}$ pin may also interfere with the reset of other parts connected to the $\overline{\text{RESET}}$ pin. Under some circumstances, it may be desirable to use an open collector circuit. See section 4.1.4.

2.14.1. Disabling The Watchdog

During program development, the watchdog can be disabled by holding the $\overline{\text{RESET}}$ pin at 2.0 to 2.5 volts. Voltages over 2.5 volts on the pin could quickly damage the part. Even at 2.5 volts, using this technique for other than

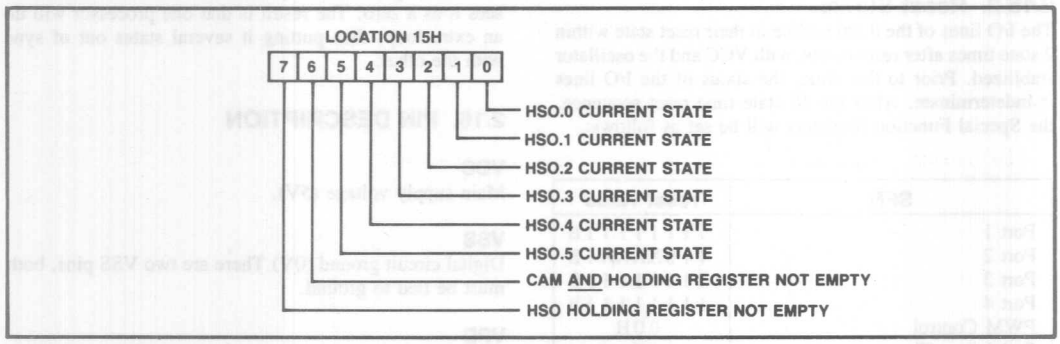


Figure 2-29. HSIO Status Register 0 (IOS0)

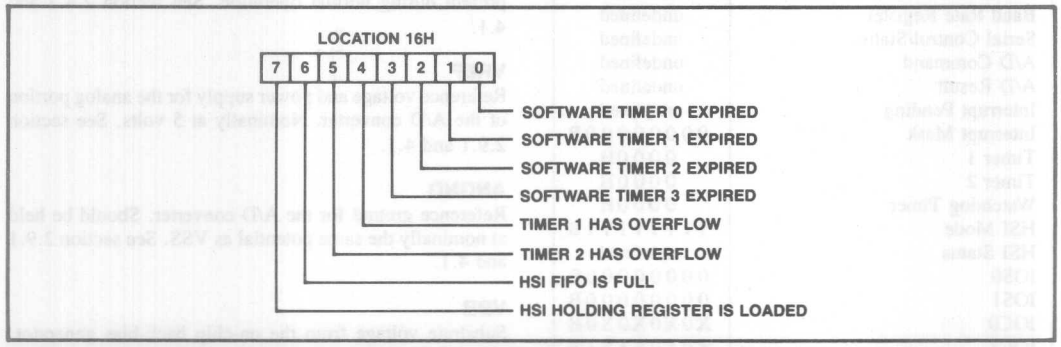


Figure 2-30. HSIO Status Register 1 (IOS1)

debugging purposes is not recommended, as it may effect long term reliability. It is further recommended that any part used in this way for more than a few minutes, not be used in production versions of products.

2.15. RESET

2.15.1. Reset Signal

As with all processors, the 8096 must be reset each time the power is turned on. To complete a reset, the RESET pin must be held active (low) for at least 2 state times after VCC, the oscillator, and the back bias generator have stabilized (~1.0 milliseconds). Then when RESET is brought high again, the 8096 executes a reset sequence that takes 10 state times. (It initializes some registers, clears the PSW and jumps to address 2080H.)

The 8096 can be reset using a capacitor, 1-shot, or any other method capable of providing a pulse of at least 2 state times longer than required for VCC and the oscillator to stabilize.

For best functionality, it is suggested that the reset pin be pulled low with an open collector device. In this way, several reset sources can be wire ored together. Remember, the RESET pin itself can be a reset source (see section 2.14). Details of hardware suggestions for reset can be found in section 4.1.4.

2.15.2. Reset Status

The I/O lines of the 8096 will be in their reset state within 2 state times after reset is low, with VCC and the oscillator stabilized. Prior to that time, the status of the I/O lines is indeterminate. After the 10 state time reset sequence, the Special Function Registers will be set as follows:

SFR	reset value
Port 1	11111111B
Port 2	110XXXX1B
Port 3	11111111B
Port 4	11111111B
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receive)	undefined
Baud Rate Register	undefined
Serial Control/Status	undefined
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00000000B
Timer 1	0000H
Timer 2	0000H
Watchdog Timer	0000H
HSI Mode	11111111B
HSI Status	undefined
IOS0	00000000B
IOS1	00000000B
IOC0	X0X0X0X0B
IOC1	X0X0XXX1B

Other conditions following a reset are:

Register	reset value
HSI FIFO	empty
HSO CAM	empty
HSO lines	00000B
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H

It is important to note that the Stack Pointer and Interrupt Pending Register are undefined, and need to be initialized in software. The Interrupts are disabled by both the mask register and PSW.9 after a reset.

2.15.3. Reset Sync Mode

The RESET line can be used to start the 8096 at an exact state time to provide for synchronization of test equipment and multiple chip systems. RESET is active low. To synchronize parts, RESET is brought high on the rising edge of XTAL1. Complete details on synchronizing parts can be found in section 4.1.5.

It is very possible that parts which start in sync may not stay that way. The best example of this would be when a "jump on I/O bit" is being used to hold the processor in a loop. If the line changes during the time it is being tested, one processor may see it as a one, while the other sees it as a zero. The result is that one processor will do an extra loop, thus putting it several states out of sync with the other.

2.16. PIN DESCRIPTION

VCC
Main supply voltage (5V).

VSS
Digital circuit ground (0V).There are two VSS pins, both must be tied to ground.

VPD
RAM standby supply voltage (5V). This voltage must be present during normal operation. See section 2.4.2 and 4.1.

VREF
Reference voltage and power supply for the analog portion of the A/D converter. Nominally at 5 volts. See section 2.9.1 and 4.1.

ANGND
Reference ground for the A/D converter. Should be held at nominally the same potential as VSS. See section 2.9.1 and 4.1.

VBB
Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01

uf capacitor (and not connected to anything else). The capacitor is not required if the A/D convertor is not being used.

XTAL1
Input of the oscillator inverter and input to the internal clock generator. See sections 2.2 and 4.1.

XTAL2
Output of the oscillator inverter. See section 2.2.

CLKOUT
Output of the internal clock generator. The frequency of CLKOUT is ⅓ the oscillator frequency. It has a 33% duty cycle. CLKOUT can drive one TTL input. See section 2.2.

$\overline{\text{RESET}}$
Reset input to the chip, also output to other circuits. Input low for at least 2 state times to reset the chip. $\overline{\text{RESET}}$ has a strong internal pullup. See section 2.15 and 4.1.

$\overline{\text{TEST}}$
Input low enables a factory test mode. The user should tie this pin to VCC for normal operation.

NMI
A low to high transition a vector to *external* memory location 0000H. Reserved for use in Intel Development systems.

INST
Output high while the address is valid during an external read indicates the read is an instruction fetch. See section 2.3.5 and 4.6.

$\overline{\text{EA}}$
Input for memory select (External Access). $\overline{\text{EA}}$ = 1 causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM. $\overline{\text{EA}}$ = 0 causes accesses to these locations to be directed to off-chip memory. $\overline{\text{EA}}$ has an internal pulldown, so it goes to 0 unless driven to 1. See section 2.3.4.

ALE
Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus. See section 2.3.5 and 4.6.

$\overline{\text{RD}}$
Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads. See section 2.3.5 and 4.6.

$\overline{\text{WR}}$
Write signal output to external memory. $\overline{\text{WR}}$ is activated only during external memory writes. See section 2.3.5 and 4.6.

$\overline{\text{BHE}}$
Bus High Enable signal output to external memory. $\overline{\text{BHE}}$ (0/1) selects/deselects the bank of memory that is connected to the high byte of the data bus. See section 2.3.5 and 4.6.

READY
The READY input is used to lengthen external memory bus cycles up to the time specified in the data sheet. It has a weak internal pullup. See section 2.3.5 and 4.6.

HSI
High impedance inputs to HSI Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. See section 2.7.

HSO
Outputs from HSO Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit. All HSO pins are capable of driving one TTL input. See section 2.8.

PORT 0/ANALOG CHANNEL
High impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. See sections 2.9 and 2.12.1.

PORT 1
Quasi-bidirectional I/O port. All pins of P1 are capable of driving one LS TTL input. See section 2.1.2.

PORT 2
Multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

Port	Function	Alternate Function	Reference section
P2.0	output	TXD (serial port transmit)	2.11.3
P2.1	input	RXD (serial port receive)	2.11.3
P2.2	input	EXTINT (external interrupt)	2.5
P2.3	input	T2CLK (Timer 2 input)	2.6.2
P2.4	input	T2RST (Timer 2 reset)	2.6.2
P2.5	output	PWM (pulse-width modulator)	2.10
P2.6	quasi-bidirectional		
P2.7	quasi-bidirectional		

The multi-functional inputs are high impedance. See section 2.1.3.

PORTS 3 AND 4

8-bit bidirectional I/O ports. These pins are shared the multiplexed address/data bus when accessing external memory, with the Port 3 pins accessing the low byte and Port 4 pins accessing the high byte. They are open drain except when being used as system bus pins. See section 2.3.5.

2.17. PIN LIST

The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, AD0-AD15, which are listed under Port 3 and Port 4.

Name	68-Pin	48-Pin
ACH0/P0.1	4	—
ACH1/P0.2	5	—
ACH2/P0.3	3	—
ACH3/P0.3	6	—
ACH4/P0.4	67	43
ACH5/P0.5	68	42
ACH6/P0.6	2	40
ACH7/P0.7	1	41
ALE	16	34
ANGND	66	44
$\overline{\text{BHE}}$	37	15
CLKOUT	13	—
$\overline{\text{EA}}$	7	39
EXTINT/P2.2	63	47
HSI.0	54	3
HSI.1	53	4
HSI.2/HSO.4	52	5
HSI.3/HSO.5	51	6
HSO.0	50	7
HSO.1	49	8
HSO.2	44	9
HSO.3	43	10
HSO.4/HSI.2	52	5
HSO.5/HSI.3	51	6
INST	15	—
NMI	7	—
PWM/P2.5	39	13
P0.0/ACH0	4	—
P0.1/ACH1	5	—
P0.2/ACH2	3	—
P0.3/ACH3	6	—
P0.4/ACH4	67	43
P0.5/ACH5	68	42
P0.6/ACH6	2	40
P0.7/ACH7	1	41
P1.0	59	—
P1.1	58	—
P1.2	57	—
P1.3	56	—
P1.4	55	—
P1.5	48	—
P1.6	47	—
P1.7	46	—

The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, TEST, T2CLK(P2.3), T2RST(P2.4).

Name	68-Pin	48-Pin
P2.0/TXD	60	2
P2.1/RXD	61	1
P2.2/EXTINT	63	47
P2.3/T2CLK	34	—
P2.4/T2RST	36	—
P2.5/PWM	39	13
P2.6	45	—
P2.7	40	—
P3.0/AD0	18	32
P3.1/AD1	19	31
P3.2/AD2	20	30
P3.3/AD3	21	29
P3.4/AD4	22	28
P3.5/AD5	23	27
P3.6/AD6	24	26
P3.7/AD7	25	25
P4.0/AD8	26	24
P4.1/AD9	27	23
P4.2/AD10	28	22
P4.3/AD11	29	21
P4.4/AD12	30	20
P4.5/AD13	31	19
P4.6/AD14	32	18
P4.7/AD15	33	17
$\overline{\text{RD}}$	17	33
READY	35	16
RESET	62	48
RXD/P2.1	61	1
TEST	14	—
TXD/P2.0	60	2
T2CLK/P2.3	34	—
T2RST/P2.4	36	—
VBB	41	12
VCC	9	38
VPD	64	46
VREF	65	45
VSS	10	11
VSS	42	37
$\overline{\text{WR}}$	38	14
XTAL1	11	36
XTAL2	12	35

MCS®-96 Software Design Information 3

CHAPTER 3
MCS®-96 SOFTWARE DESIGN INFORMATION

3.0. INTRODUCTION

This section provides information which will primarily interest those who must write programs to execute in the 8096. Several other sources of information are currently available which will also be of interest:

MCS®-96 MACRO ASSEMBLER USER'S GUIDE
Order Number 122048-001

MCS-96 UTILITIES USER'S GUIDE
Order Number 122049-001

MCS-96 MACRO ASSEMBLER AND UTILITIES POCKET REFERENCE
Order Number 122050-001

Throughout this chapter short segments of code are used to illustrate the operation of the device. For these sections it has been assumed that a set of temporary registers have been predeclared. The names of these registers have been chosen as follows:

AX, BX, CX, and DX are 16 bit registers.

AL is the low byte of AX, AH is the high byte.

BL is the low byte of BX

CL is the low byte of CX

DL is the low byte of DX

These are the same as the names for the general data registers used in 8086. It is important to note, however, that in the 8096, these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte region within onboard register file.

3.1. OPERAND TYPES

The MCS®-96 architecture provides support for a variety of data types which are likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language will be used where appropriate. To avoid confusion the name of an operand type will be capitalized. A "BYTE" is an unsigned eight bit variable; a "byte" is an eight bit unit of data of any type.

3.1.1. Bytes

BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within BYTES are labeled from 0 to 7 with 0 being the least significant bit. There are no alignment restrictions for BYTES so they may be placed anywhere in the MCS-96 address space.

3.1.2. Words

WORDS are unsigned 16-bit variables which can take on the values between 0 and 65535. Arithmetic and relational operators can be applied to WORD operands but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte.

3.1.3. Short-Integers

SHORT-INTEGERS are 8-bit signed variables which can take on the values between −128 and +127. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERS so they may be placed anywhere in the MCS-96 address space.

3.1.4. Integers

INTEGERS are 16-bit signed variables which can take on the values between −32,768 and 32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

3.1.5. Bits

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8096 provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

3.1.6. Double-Words

DOUBLE-WORDS are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 8096 and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with INTEL provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in section 3.5.

3.1.7. Long-Integers

LONG-INTEGERS are 32-bit signed variables which can take on the values between −2,147,483,648 and 2,147,483,647. The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 8096 and be aligned at an address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte.

LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in section 3.5.

3.2. OPERAND ADDRESSING

Operands are accessed within the address space of the 8096 with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be described as they are seen through the assembly language. The six basic addressing modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

3.2.1. Register-direct References

The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and register address must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in the calculation.

```
Examples
   ADD     AX,BX,CX      ; AX: = BX + CX
   MUL     AX,BX         ; AX: = AX*BX
   INCB    CL            ; CL: = CL + 1
```

3.2.2. Indirect References

The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of the 8096, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

```
Examples
   LD      AX,[AX]       ; AX: = MEM _ WORD(AX)
   ADDB    AL,BL,[CX]    ; AL: = BL + MEM _ BYTE(CX)
   POP     [AX]          ; MEM _ WORD(AX) : = MEM _ WORD(SP); SP: = SP + 2
```

3.2.3. Indirect with Auto-increment References

This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT-INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect address able will be incremented by two.

```
Examples
   LD      AX,[BX]+      ; AX: = MEM _ WORD(BX); BX: = BX + 2
   ADD     AL,BL,[CX]+   ; AL: = AL + BL + MEM _ BYTE(CX); CX: = CX + 1
   PUSH    [AX]+         ; SP: = SP − 2;
                         ;    MEM _ WORD(SP): = MEM _ WORD(AX)
                         ;    AX: = AX + 2
```

3.2.4. Immediate References

This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER operands the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining operand(s) must be register-direct references.

```
Examples
    ADD   AX,#340  ; AX: = AX + 340
    PUSH  #1234H   ; SP: = SP - 2; MEM _ WORD(SP): = 1234H
    DIVB  AX,#10   ; AL: = AX/10; AH: = AX MOD 10
```

3.2.5. Short-indexed References

In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which is assumed to contain an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the operand which will take part in the calculation. Since the eight bit field is sign-extended the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

```
Examples
    LD    AX,12[BX]    ; AX: = MEM _ WORD(BX + 12)
    MULB  AX,BL,3[CX]  ; AX: = BL*MEM _ BYTE(CX + 3)
```

3.2.6. Long-indexed References

This addressing mode is like the short-indexed mode except that a *16-bit* field is taken from the instruction and added to the WORD variable to form the address of the operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must to register-direct references.

```
Examples
    AND   AX,BX,TABLE[CX]   ; AX: = BX AND MEM _ WORD(TABLE + CX)
    ST    AX,TABLE[BX]      ; MEM _ WORD(TABLE + BX) : = AX
    ADDB  AL,BL,LOOKUP[CX]  ; AL: = BL + MEM _ BYTE(LOOKUP + CX)
```

3.2.7. ZERO Register Addressing

The first two bytes in the register file are fixed at zero by the 8096 hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

```
Examples
    ADD   AX,1234[0]  ; AX: = AX + MEM _ WORD(1234)
    POP   5678[0]     ; MEM _ WORD(5678): = MEM _ WORD(SP)
                      ;   SP: = SP + 2
```

3.2.8. Stack Pointer Register Addressing

The system stack pointer in the 8096 can be accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

```
Examples
    PUSH  [SP]      ; DUPLICATE TOP _ OF _ STACK
    LD    AX,2[SP]  ; AX: = NEXT _ TO _ TOP
```

3.2.9. Assembly Language Addressing Modes

The 8096 assembly language simplifies the choice of addressing modes to be used in several respects:

Direct Addressing. The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

Immediate Addressing. The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

The use of these features of the assembly language simplifies the programming task and should be used wherever possible.

3.3 PROGRAM STATUS WORD

The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. The format of the PSW is shown in figure 3-1. The information in the PSW can be broken down into two basic categories; interrupt control and condition flags. The PSW can be saved in the system stack with a single operation (PUSHF) and restored in a like manner (POPF).

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

Figure 3-1. PSW Register

3.3.1. Interrupt Flags

The lower eight bits of the PSW are used to individually mask the various sources of interrupt to the 8096. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. These mask bits can be accessed as an eight bit byte (INT_MASK — address 8) in the on-board register file. Bit 9 in the PSW is the global interrupt enable. If this bit is cleared then all interrupts will be locked out except for the Non Maskable Interrupt (NMI). Note that the various interrupts are collected in the INT _ PENDING register even if they are locked out. Execution of the corresponding service routines will procede according to their priority when they become enabled. Further information on the interrupt structure of the 8096 can be found in sections 2.5 and 3.6.

3.3.2. Condition Flags

The remaining bits in the PSW are set as side effects of instruction execution and can be tested by the conditional jump instructions.

Z. The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

N. The N (Negative) flag is set to indicate that the operation generated a negative result. Note that the N flag will be set to the algebraically correct state even if the calculation overflows.

V. The V (oVerflow) flag is set to indicate that the operation generated a result which is outside the range that can be expressed in the destination data type.

VT. the VT (oVerflow Trap) flag is set whenever the V flag is set but can only be cleared an instruction which explicitly operates on it such as the CLRVT or JVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

C. The C (Carry) flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation or the state of the last bit shifted out of the operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C = 0).

ST. The ST (STicky bit) set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

```
MULUB   AX,CL,DL    ; AX: = CL*DL
SHR     AX,#4       ; Shift right 4 places
```

If the C flag is set after the shift it indicates that the bits shifted off the end of operand were greater-than or equal-to one half the least significant bit (LSB) of the result. If the C flag is clear after the shift it indicates that the bits shifted off the end of the operand were less than half the LSB of the result. Without the ST flag, the rounding decision must be made on the basis of this information alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision:

C ST	Value of the bits shifted off
0 0	Value = 0
0 1	0 < Value < ½ LSB
1 0	Value = ½ LSB
1 1	Value > ½ LSB

Figure 3-2. Rounding Alternatives

Imprecise rounding can be a major source of error in a numerical calculation; use of the ST flag improves the options available to the programmer.

3.4 INSTRUCTION SET

The MCS-96 instruction set contains a full set of arithmetic and logical operations for the 8-bit data types BYTE and SHORT INTEGER and for the 16-bit data types WORD and INTEGER. The DOUBLE-WORD and LONG data types (32 bits) are supported for the products of 16 by 16 multiplies and the dividends of 32 by 16 divides and for shift operations. The remaining operations on 32 bit variables can be implemented by combinations of 16 bit operations. As an example the sequence:

```
ADD     AX,CX
ADDC    BX,DX
```

performs a 32 bit addition, and the sequence

```
SUB     AX,CX
SUBC    BX,DX
```

performs a 32 bit subtraction. Operations on REAL (i.e. floating point) variables are not supported directly by the hardware but are supported by the floating point library for the 8096 (FPAL-96) which implements a single precision subset of the proposed IEEE standard for floating point operations. The performance of this software is significantly improved by the 8096 NORML instruction which normalizes a 32-bit variable and by the existance of the ST flag in the PSW.

In addition to the operations on the various data types, the 8096 supports conversions between these types. LDBZE (load byte zero extended) converts a BYTE to a WORD and LDBSE (load byte sign extended) converts a SHORT-INTEGER into an INTEGER. WORDS can be converted to DOUBLE-WORDS by simply clearing the upper WORD of the DOUBLE-WORD (CLR) and INTEGERS can be converted to LONGS with the EXT (sign extend) instruction.

Tables 3-1 and 3-2 summarize the operation of each of the instructions and Tables 3-3 and 3-4 give the opcode, byte count, and timing information for each of the instructions.

Table 3-1. Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D+A	√	√	√	√	√	—	
ADD/ADDB	3	D ← B+A	√	√	√	√	√	—	
ADDC/ADDCB	2	D ← D+A+C	√	√	√	√	√	—	
SUB/SUBB	2	D ← D − A	√	√	√	√	√	—	
SUB/SUBB	3	D ← B − A	√	√	√	√	√	—	
SUBC/SUBCB	2	D ← D − A + C − 1	√	√	√	√	√	—	
CMP/CMPB	2	D − A	√	√	√	√	√	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	√	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	√	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	√	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	√	3
DIV/DIVU	2	D ← (D, D + 2)/A D + 2 remainder	—	—	—	√	√	—	2
DIVB/DIVUB	3	D ← (D, D + 1)/A D + 1 remainder	—	—	—	√	√	—	3
AND/ANDB	2	D ← D and A	√	√	0	0	—	—	
AND/ANDB	3	D ← B and A	√	√	0	0	—	—	
OR/ORB	2	D ← D or A	√	√	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	√	√	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3,4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3,4
PUSH	1	SP ← SP − 2; (SP) A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2	√	√	√	√	√	√	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
INDJMP	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J(conditional)	1	PC ← PC + 8-bit offset	—	—	—	—	—	—	5
JC		Jump if C = 1	—	—	—	—	—	—	5
JNC		Jump if C = 0	—	—	—	—	—	—	5

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Table 3-2. Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JE		Jump if Z = 1	—	—	—	—	—	—	5
JNE		Jump if Z = 0	—	—	—	—	—	—	5
JGE		Jump if N = 0	—	—	—	—	—	—	5
JLT		Jump if N = 1	—	—	—	—	—	—	5
JGT		Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE		Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH		Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH		Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV		Jump if V = 1	—	—	—	—	—	—	5
JNV		Jump if V = 0	—	—	—	—	—	—	5
JVT		Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT		Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST		Jump if ST = 1	—	—	—	—	—	—	5
JNST		Jump if ST = 0	—	—	—	—	—	—	5
JBS		Jump if ST = 1	—	—	—	—	—	—	5,6
JBC		Jump if Specified Bit = 0	—	—	—	—	—	—	5,6
DJNZ	1	D ← D − 1; if D ≠ then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	√	√	√	√	√	—	
NEG/NEGB	1	D ← 0 − D	√	√	√	√	√	—	
INC/INCB	1	D ← D + 1	√	√	√	√	√	—	
EXT	1	D ← D; D + 2 ← Sign (D)	√	√	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	√	√	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	√	√	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	1	C ← msb — — — — — lsb ← 0	√	√	√	√	√	—	7
SHR/SHRB/SHRL	1	0 → msb — — — — — lsb → C	√	√	√	0	—	√	7
SHRA/SHRAB/SHRAL	1	msb → msb — — — — — lsb → C	√	√	√	0	—	√	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts	—	—	—	—	—	—	
EI	0	Enable All Interrupts	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Normalize (See sec 3.13.66)	√	1	—	—	—	—	7

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.

Table 3-3. Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊛ NORMAL			INDIRECT⊛ AUTO-INC.		INDEXED⊛ SHORT			INDEXED⊛ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	27/32	3	28/33	8F	4	27/32	5	28/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	19/24	3	20/25	9F	4	19/24	5	20/25
DIV	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
DIVB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29

Notes:

⊛ Long indexed and Indirect + instructions have identical opocodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

Table 3-3. Continued

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT③ NORMAL			INDIRECT③ AUTO-INC.		INDEXED③ SHORT			INDEXED③ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	——	C2	3	7/13	3	8/14	C3	4	7/13	5	8/14
STB	2	C4	3	4	—	—	——	C6	3	7/13	3	8/14	C7	4	7/13	5	8/14
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	——	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	——	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
INDJMP	E3	2	8	RET	F0	1	12/16⑤

Notes:

① Number of state times shown for internal/external operands.

③ This instruction is generated by the assembler when a branch indirect (BR [Rx]) instruction is reached.

④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.

⑤ State times for stack located internal/external.

Table 3-4. CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

DJNZ	OPCODE E0;	3 BYTES;	5/9 STATE TIMES (NOT TAKEN/TAKEN)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR	WORD		INSTR	BYTE		INSTR	DBL WD		
MNEMONIC	OP	B	MNEMONIC	OP	B	MNEMONIC	OP	B	STATE TIMES
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT⑦
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT⑦
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT⑦

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST	FF	1	16	SKIP	00	2	4

NORMALIZE

NORML	0F	3	11 + 1 PER SHIFT

Notes:

⑥ This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.

⑦ Execution will take at least 8 states, even for 0 shift.

3.5. SOFTWARE STANDARDS AND CONVENTIONS

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programing language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

3.5.1. Register Utilization

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

3.5.2. Addressing 32-bit Operands

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

3.5.3. Subroutine Linkage

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed into the stack as two 16 bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```
example_procedure: PROCEDURE (paraml,param2,param3);
  DECLARE paraml BYTE,
          param2 DWORD,
          param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:

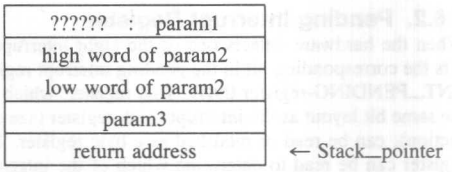


Figure 3-3. Stack Image

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8, 16 or 32 bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

a). Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.

b). Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.

c). The Program Status Word (PSW-see section 3.3) is not saved and restored by procedures so the calling code must assumed that the condition flags (Z,N,V,VT,C, and ST) are modified by the procedure.

d). Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

3.6. USING THE INTERRUPT SYSTEM

Processing interrupts is an integral part of almost any control application. The 8096 allows the program to manage interrupt servicing in an efficient and flexible manner. Software running in the 8096 exerts control over the interrupt hardware at several levels.

3.6.1. Global Lockout

The processing of interrupts can be enabled or disabled by setting or clearing the I bit in the PSW. This is accomplished by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts; interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

3.6.2. Pending Interrupt Register

When the hardware detects one of the eight interrupts it sets the corresponding bit in the pending interrupt register (INT_PENDING-register 09H). This register, which has the same bit layout as the interrupt mask register (see next section), can be read or modified as a byte register. This register can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT_PENDING register should ensure that the entire operation is indivisible. The easiest way of doing this is to use the logical instructions in the two or three operand format, as examples:

```
ANDB    INT_PENDING,#11111101B
ORB     ; Clears the A/D interrupt
        INT_PENDING,#00000010B
        Sets the A/D interrupt
```

If the required modification to INT_PENDING cannot be accomplished with one instruction then a critical region should be established and the INT_PENDING register modified from within this region (see section 3.6.5).

3.6.3. Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask register (INT_MASK-register 08H). The format of this register is shown in figure 3-4.

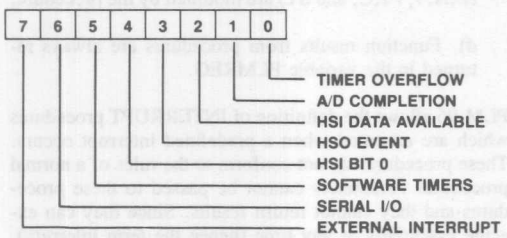


Figure 3-4. Interrupt Mask Register

The INT_MASK register can be read or written as a byte register. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The individual masks act like the global lockout in that they only control the servicing of the interrupt; the hardware will save any interrupts that occur in the pending register even if the interrupt mask bit is cleared. The INT_MASK register also can be accessed as the lower eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT_MASK register as well as the global interrupt lockout and the arithmetic flags.

3.6.4. Interrupt Vectors

The 8096 has eight sources of hardware interrupt, each with its own priority and interrupt vector location. Table 3-6 shows the interrupt sources, their priority, and their vector locations. See section 2.5 for a discussion of the various interrupt sources.

Table 3-5. Interrupt Vector Information

Source	Priority	Vector
Timer Overflow	0-Lowest	2000H
A/D Completion	1	2002H
HSI Data Available	2	2004H
HSO Excution	3	2006H
HSI.O	4	2008H
Software timers	5	200AH
Serial I/O	6	200CH
External Interrupt	7-Highest	200EH

The programmer must initialize the interrupt vector table with the starting addresses of the appropriate interrupt service routine. It would be a good idea to vector any interrupts that are not used in the system to an error handling routine.

The priorities given in the table give the *hardware* enforced priorities for these interrupts. This priority controls the order in which pending interrupts are passed to the software via interrupt-calls. The software can implement its own priority structure by controlling the mask register (INT _ MASK-register 08H). To see how this is done consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```
serial_io_isr:
    PUSHF           ; Save the PSW
                    (Includes INT_MASK)
    LD      INT_MASK,#00000100B
    EI              ; Enable interrupts again
    ;
    ;
    ;               } Service the interrupt
    ;
    ;
    ;
    POPF            ; Restore the PSW
    RET
```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial_io_isr and the interrupt be enabled for this routine to execute.

There is an interesting chain of instruction side-effects which makes this (or any other) 8096 interrupt service routine execute properly:

a). After the hardware decides to process an interrupt it generates and executes a special interrupt-call instruction which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts the first instruction of the interrupt service routine will execute.

b). The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT_MASK register and the global enable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and by the time the LD instruction starts all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT_MASK instruction.

c). The LD INT_MASK instruction enables those interrupts that the programmer chooses to allow to interrupt the serial I/O interrupt service routine. In this example only the HSI data available interrupt will be allowed to do this but any interrupt or combination of interrupts could be enabled at this point, even the serial interrupt. It is the loading of the INT_MASK register which allows the software to establish its own priorities for interrupt servicing independently from those that the hardware enforces.

d). The EI instruction reenables the processing of interrupts.

e). The actual interrupt service routine executes within the priority structure established by the software.

f). At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt-call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenable an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency.

Notice that the "preamble" and exit code for the interrupt service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the on-board register file. The availability of some 230 bytes of register storage makes this quite practical.

3.6.5. Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

```
LDB     AL,INT_PENDING
ANDB    AL,#bit_mask
STB     AL,INT_PENDING
```

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT_PENDING register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT_PENDING contains 00001111B and after the LDB instruction so does AL. IF the HSI interrupt service routine executes at this point then INT_PENDING will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT_PENDING to 00000111B. It should be 00000011B. This code sequence has managed to generate a false HSO interrupt! The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 8096 allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction:

```
ANDB    INT_PENDING,#bit _ mask
```

Instructions are indivisible so mutual exclusion is ensured in this case. For more complex situations, such a simple solution is not available and the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears the interrupt enable flags. The region can then be terminated with a POPF instruction which returns the interrupt enable to the state it was in before the code sequence. It

should be noted that some system configurations might require more protection to form a critical region. An example is a system in which more than one processor has access to a common resource such as memory or external I/O devices.

3.7. I/O PROGRAMMING CONSIDERATIONS

The on-board I/O devices are, for the most part, simple to program. There are some areas of potential confusion which need to be addressed:

3.7.1. Programming the I/O Ports

Some of the on-board I/O ports can be used as both input and output pins (e.g. Port 1) When the processor writes to the pins of these ports it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to that pin, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with relatively high impedance pull-up device which can be easily driven down by the device driving the input. If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port. Consider using P1.0 as an input and then trying to toggle P1.1 as an output:

```
ORB    IOPORT1,#00000001B    ; Set P1.0 for input
XORB   IOPORT1,#00000010B    ; Complement P1.1
```

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven high by the 8096 it is possible that it is being held low externally. This typically happens when the port pin is used to drive the base of an NPN transistor which in turn drives whatever there is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's Vbe above ground, typically 0.7 volts. The 8096 will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin and it will not toggle. The second problem, which is related to the first one, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction then the XORB will write a zero to P1.0 and it will no longer be useable as an input. The first problem can best be solved by the external driver design. A series resistor between the port pin and the base of the transistor often works. The second problem can be solved in the software fairly easily:

```
LDB    AL,IOPORT1
XORB   AL,#010B
ORB    AL,#001B
STB    AL,IOPORT1
```

A software solution to both problems is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and then copy it to the port.

3.7.2. Reading the I/O Status Register 1

This status register contains a collection of status flags which relate to the timer and high speed I/O functions (see section 2.12.5). It can be accessed as register 16H in the on-board register file. The layout of this register is shown in figure 3-5.

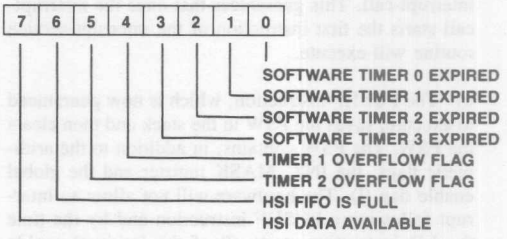


Figure 3-5. I/O Status Register 1

Whenever the processor reads this register all of the time-related flags (bits 5 through 0) are cleared. This applies not only to explicit reads such as:

```
LD     AL,IOS1
```

but also to implicit reads such as:

```
JB     IOS1.3,somewhere_else
```

which jumps to somewhere_else if bit 3 of IOS1 is set. In most cases this situation can best be handled by having a byte in the register file which is used to maintain an image of lower five bits of the register. Any time a hardware timer interrupt or a HSO software timer interrupt occurs the byte can be updated:

```
ORB    IOS1_image,IOS1
```

leaving IOS1_image containing all the flags that were set before plus all the new flags that were read and cleared from IOS1. Any other routine which needs to sample the flags can safely check IOS1_image. Note that if these routines need to clear the flags that they have acted on then the modification of IOS1_image must be done from inside a critical region (see section 3.6.5).

3.7.3. Sending Commands to the HSO Unit

Commands are sent to the HSO unit via a byte and then a word write operation:

```
LDB    HSO_COMMAND,#what_to_do
ADD    HSO_TIME,TIMER1,#when_to_do_it
```

The command is actually accepted when the HSO_TIME register is written. It is important to ensure that this code piece is not interrupted by any interrupt service routine

which might also send a command to the HSO unit. If this happens the HSO will know when to do it but not know what to do when it's time to do it. In many systems this becomes a null problem because HSO commands are only issued from one place in the code. If this is not the case then a critical region must be established and the two instructions executed from within this region (see section 3.6.5).

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM for this to occur. Flags are available in IOS0 which indicate the holding register is empty (IOS0.7) or that both the holding register is empty *and* the CAM is not full (IOS0.6). The programmer should carefully decide which of these two flags is the best to use for each application.

It is possible to enter commands into the CAM which never execute. This occurs if TIMER2 has been set up as a variable modulo counter and a command is entered with a time tag referenced to TIMER2 which has a value that TIMER2 never reaches. The inaccessible command will never execute and continue to take up room in the CAM until either the system is reset or the program allows TIMER2 increment up to the value stored in the time tag. Note that commands cannot be flushed from the CAM without being executed but that they can be cancelled. This is accomplished by setting the opposite command in the CAM to execute at the same time tag as the command to be cancelled. If, as an example, a command has been issued to set HSO.1 when TIMER1 = 1234 then entering a second command which clears HSO.1 when TIMER1 = 1234 will result in a no-operation on HSO.1. Both commands will remain in the CAM until TIMER1 = 1234.

3.7.4. High Speed I/O Interrupts

The HSO unit can generate two types of interrupts. The HSO execution interrupt (vector = (2006H)) is generated (if enabled) for HSO commands which operate on one of the six HSO pins. The other HSO interrupt is the Software Timer interrupt (vector = (200AH)) which is generated (if enabled) for any other HSO command (e.g. triggering the A/D, resetting Timer2 or generating a software time delay).

There are also two interrupts associated with the HSI unit. The HSI data available interrupt (vector = (2004H)) is generated if there is data in the HSI FIFO that the program should read. The other HSI related interrupt is the HSI.0 interrupt which occurs whenever High Speed Input pin 0 makes a zero-to-one transition. This interrupt will become pending in the INT_PENDING register even if the HSI unit is programmed to ignore changes on HSI.0 or look for a one-to-zero transition.

3.7.5. Accessing Register Mapped I/O

The on-board I/O devices such as the serial port or the A/D converter are controlled as register mapped I/O. This allows convenient and efficient I/O processing. The implementation of the current members of the MCS-96 family place some restrictions on how these registers can be accessed. While these restrictions are not severe, the programmer must be aware of them. A complete listing of these registers is shown in figure 2-7 and 2-8. The restrictions are as follows:

a). TIMER0, TIMER1 and HSI_TIME are *word read only*. They cannot be read as bytes or written to in any format.

b). HSO_TIME is *word write only*. It cannot be written to as individual bytes or read in any format.

c). R0 (the ZERO register) is byte or word read or write but writing to it will not change its value.

d). All of the other I/O registers can be accessed only as bytes. This applies even to the AD_RESULT which is logically a word operand.

3.8. EXAMPLE-1 PROGRAMMING THE SERIAL I/O CHANNEL

```
MCS-96 MACRO ASSEMBLER    SERIAL PORT DEMO PROGRAM

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:SPX.SRC
OBJECT FILE: :F1:SPX.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1   $TITLE('SERIAL PORT DEMO PROGRAM')
                                    2   $PAGELENGTH(95)
                                    3
                                    4   ;       This program initializes the serial port and echos any
                                    5   ;       character sent to it.
                                    6
                                    7
     000E                           8   BAUD_REG        equ     0EH
     0011                           9   SPCON           equ     11H
     0011                          10   SPSTAT          equ     11H
     0016                          11   IOC1            equ     16H
     0015                          12   IOC0            equ     15H
     0007                          13   SBUF            equ     07H
     0009                          14   INT_PENDING     equ     09H
     0018                          15   SP              equ     18H
                                   16
                                   17
     0000                          18   rseg
                                   19
     0000                          20           CHR:    dsb     1
     0001                          21           TEMP0:  dsb     1
     0002                          22           TEMP1:  dsb     1
     0003                          23           RCV_FLAG:       dsb     1
                                   24
                                   25
     0000                          26   cseg
                                   27
     0000 A1B00018                 28           LD      SP, #0B0H
                                   29
     0004 B12016                   30           LDB     IOC1, #00100000B            ; Set P2.0 to TXD
                                   31
                                   32           ; Baud rate = input frequency / (64*baud_val)
                                   33           ; baud_val  = (input frequency/64) / baud rate
                                   34
                                   35
       0027                        36   baud_val        equ     39      ; 2400 baud at 6.0 MHz
                                   37
       0080                        38   BAUD_HIGH       equ     ((baud_val-1)/256) OR 80H   ; Set MSB to 1
       0026                        39   BAUD_LOW        equ     (baud_val-1) MOD 256
                                   40
                                   41
     0007 B1260E                   42           LDB     BAUD_REG, #BAUD_LOW
     000A B1800E                   43           LDB     BAUD_REG, #BAUD_HIGH
                                   44
     000D B14911                   45           LDB     SPCON, #01001001B       ; Enable receiver, Mode 1
                                   46
                                   47                   ; The serial port is now initialized
                                   48
                                   49
     0010 C40007          R        50           STB     SBUF, CHR               ; Clear serial Port
     0013 B12001          R        51           LDB     TEMP0, #00100000B       ; Set TI-temp
                                   52
     0016 3609FD                   53   wait:   JBC     INT_PENDING, 6, wait    ; Wait for pending bit to be set
     0019 71BF09                   54           ANDB    INT_PENDING, #10111111B ; Clear pending bit
                                   55
     001C 901101          R        56           ORB     TEMP0, SPCON            ; Put SPCON into temp register
                                   57                                           ; This is necessary becase reading
                                   58                                           ; SPCON clears TI and RI
                                   59
     001F                          60   get_byte:
     001F 360109          R        61           JBC     TEMP0, 6, put_byte      ; If RI-temp is not set
     0022 C40007          R        62           STB     SBUF, CHR               ; Store byte
     0025 71BF01          R        63           ANDB    TEMP0, #10111111B       ; CLR RI-temp
     0028 B1FF03          R        64           LDB     RCV_FLAG, #0FFH         ; Set bit-received flag
                                   65
     002B                          66   put_byte:
     002B 30030C          R        67           JBC     RCV_FLAG, 0, continue   ; If receive flag is cleared
     002E 350109          R        68           JBC     TEMP0, 5, continue      ; If TI was not set
     0031 B00007          R        69           LDB     SBUF, CHR               ; Send byte
     0034 71DF01          R        70           ANDB    TEMP0, #11011111B       ; CLR TI-temp
     0037 B10003          R        71           LDB     RCV_FLAG, #00           ; Clear bit-received flag
                                   72
     003A                          73   continue:
     003A 27DA                     74           BR      wait
                                   75
     003C                          76           END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

3.9. EXAMPLE-2 GENERATING A PWM WITH THE HSO UNIT

```
MCS-96 MACRO ASSEMBLER    HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0
SOURCE FILE: :F1:HSO2X.SRC
OBJECT FILE: :F1:HSO2X.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT          LINE      SOURCE STATEMENT
                               $TITLE ('HSO EXAMPLE PROGRAM FOR PWM OUTPUTS')
                               $PAGELENGTH(95)

                               ; This program will provide 4 PWM outputs on HSO pins 0-3
                               ; The input parameters passed to the program are:
                               ;
                               ;          HSO_ON_N     HSO on time for pin N
                               ;          HSO_OFF_N    HSO off time for pin N

                               ;    Where:  Times are in timer1 cycles
                               ;            N takes values from 0 to 3

                               ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    0000                       dseg

    0000                               D_STAT:         DSB     1
                                       extrn   HSO_ON_0 :word , HSO_OFF_0 :word
                                       extrn   HSO_ON_1 :word , HSO_OFF_1 :word
                                       extrn   HSO_ON_2 :word , HSO_OFF_2 :word
                                       extrn   HSO_ON_3 :word , HSO_OFF_3 :word
                                       extrn   HSO_TIME :word , HSO_COMMAND :byte
                                       extrn   TIMER1   :word , IOS0      :byte
                                       extrn   SP       :word


    0000                       rseg

                                       public  OLD_STAT
    0000                               OLD_STAT:       dsb     1
    0001                               NEW_STAT:       dsb     1


    0000                       cseg

                                       PUBLIC  wait


    0000 3E00FD          E     wait:   JBS     IOS0, 6, wait              ; Loop until HSO holding register
    0003 FD                            NOP                                ; is empty
    0004 FD                            NOP
    0005 C701000000      E             STB     IOS0, D_STAT               ; Load byte to external RAM

                                               ; For opperation with interrupts 'store_stat:' would be the
                                               ; entry point of the routine.
                                               ; Note that a DI or PUSHF might have to be added.

    000A                       store_stat:
    000A 510F0001        E             ANDB    NEW_STAT, IOS0, #0FH       ; Store new status of HSO
    000E 980100          R             CMPB    OLD_STAT, NEW_STAT
    0011 DFED                          JE      wait                       ; If status hasn't changed
    0013 940100          R             XORB    OLD_STAT, NEW_STAT


    0016                       check_0:
    0016 300017          R             JBC     OLD_STAT, 0, check_1       ; Jump if OLD_STAT(0)=NEW_STAT(0)
    0019 38010B          R             JBS     NEW_STAT, 0, set_off_0

    001C                       set_on_0:
    001C B13000          E             LDB     HSO_COMMAND, #00110000B    ; Set HSO for timer1, set pin 0
    001F 470100000000    E             ADD     HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = Timer1 value
    0025 2009                          BR      check_1                    ;   + Time for pin to be low

    0027                       set_off_0:
    0027 B11000          E             LDB     HSO_COMMAND, #00010000B    ; Set HSO for timer1, clear pin 0
    002A 470100000000    E             ADD     HSO_TIME, TIMER1, HSO_ON_0 ; Time to clear pin = Timer1 value
                                                                          ;   + Time for pin to be high

    0030                       check_1:
    0030 310017          R             JBC     OLD_STAT, 1, check_2       ; Jump if OLD_STAT(1)=NEW_STAT(1)
    0033 39010B          R             JBS     NEW_STAT, 1, set_off_1

    0036                       set_on_1:
    0036 B13100          E             LDB     HSO_COMMAND, #00110001B    ; Set HSO for timer1, set pin 1
    0039 470100000000    E             ADD     HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = Timer1 value
    003F 2009                          BR      check_2                    ;   + Time for pin to be low

    0041                       set_off_1:
    0041 B11100          E             LDB     HSO_COMMAND, #00010001B    ; Set HSO for timer1, clear pin 1
    0044 470100000000    E             ADD     HSO_TIME, TIMER1, HSO_ON_1 ; Time to clear pin = Timer1 value
                                                                          ;   + Time for pin to be high

                               $EJECT
```

```
MCS-96 MACRO ASSEMBLER    HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

ERR LOC   OBJECT                LINE        SOURCE STATEMENT

                                  85
    004A                          86    check_2:
    004A 320017           R       87          JBC     OLD_STAT, 2, check_3          ; Jump if OLD_STAT(2)=NEW_STAT(2)
    004D 3A010B           R       88          JBS     NEW_STAT, 2, set_off_2
                                  89
    0050                          90    set_on_2:
    0050 B13200           E       91          LDB     HSO_COMMAND, #00110010B       ; Set HSO for timer1, set pin 2
    0053 470100000000     E       92          ADD     HSO_TIME, TIMER1, HSO_OFF_2   ; Time to set pin = Timer1 value
    0059 2009                     93          BR      check_3                       ;  + Time for pin to be low
                                  94
    005B                          95    set_off_2:
    005B B11200           E       96          LDB     HSO_COMMAND, #00010010B       ; Set HSO for timer1, clear pin 2
    005E 470100000000     E       97          ADD     HSO_TIME, TIMER1, HSO_ON_2    ; Time to clear pin = Timer1 value
                                  98                                                ;  + Time for pin to be high
                                  99
                                 100
    0064                         101    check_3:
    0064 330017           R      102          JBC     OLD_STAT, 3, check_done       ; Jump if OLD_STAT(3)=NEW_STAT(3)
    0067 3B010B           R      103          JBS     NEW_STAT, 3, set_off_3
                                 104
    006A                         105    set_on_3:
    006A B13300           E      106          LDB     HSO_COMMAND, #00110011B       ; Set HSO for timer1, set pin 3
    006D 470100000000     E      107          ADD     HSO_TIME, TIMER1, HSO_OFF_3   ; Time to set pin = Timer1 value
    0073 2009                    108          BR      check_done                    ;  + Time for pin to be low
                                 109
    0075                         110    set_off_3:
    0075 B11300           E      111          LDB     HSO_COMMAND, #00010011B       ; Set HSO for timer1, clear pin 3
    0078 470100000000     E      112          ADD     HSO_TIME, TIMER1, HSO_ON_3    ; Time to clear pin = Timer1 value
                                 113                                                ;  + Time for pin to be high
                                 114
                                 115
    007E                         116    check_done:
    007E B00100           R      117          LDB     OLD_STAT, NEW_STAT            ; Store current status and
                                 118                                                ; wait for interrupt flag
    0081 F0                      119          RET
                                 120
                                 121
    0082                         122          END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

3.10. EXAMPLE-3 MEASURING PULSES WITH THE HSI UNIT

```
MCS-96 MACRO ASSEMBLER    MEASURING PULSES USING THE HSI UNIT

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:PULSEX.SRC
OBJECT FILE: :F1:PULSEX.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                  1   $TITLE('MEASURING PULSES USING THE HSI UNIT')
                                  2   $PAGELENGTH(95)
                                  3
                                  4   ;       This program measures pulsewidths in TIMER1 cycles
                                  5   ;       and retuns the values in external RAM.
                                  6
                                  7
     0018                         8   SP              equ     18H
     0003                         9   HSI_MODE        equ     03H
     0006                        10   HSI_STATUS      equ     06H
     0004                        11   HSI_TIME        equ     04H
     000A                        12   TIMER1          equ     0AH
     0015                        13   IOC0            equ     15H
     0016                        14   IOS1            equ     16H
                                 15
                                 16
0000                             17   rseg
                                 18
0000                             19           HIGH_TIME:      dsw     1
0002                             20           LOW_TIME:       dsw     1
0004                             21           PERIOD:         dsw     1
0006                             22           HI_EDGE:        dsw     1
0008                             23           LO_EDGE:        dsw     1
                                 24
                                 25
000A                             26           AX:     dsw     1
  000A                           27           AL      equ     AX      :byte
  000B                           28           AH      equ     (AX+1)  :byte
                                 29
000C                             30           BX:     dsw     1
  000C                           31           BL      equ     BX      :byte
  000D                           32           BU      equ     (BX+1)  :byte   ; Note that 'BH' is an opcode so it
                                 33                                           ; can't be used as a label
                                 34
0000                             35   cseg
                                 36
0000 A1C00018                    37           LD      SP, #0C0H
0004 B10115                      38           LDB     IOC0, #00000001B        ; Enable HSI 0
0007 B10F03                      39           LDB     HSI_MODE, #00001111B    ; HSI 0 look for either edge
                                 40
000A 44020004          R         41   wait:   ADD     PERIOD, HIGH_TIME, LOW_TIME
                                 42
000E 3716F9                      43           JBC     IOS1, 7, wait   ; Wait while no pulse is entered
                                 44
0011 B0060A            R         45           LDB     AL, HSI_STATUS          ; Load status; Note that reading
                                 46                                           ;   HSI_TIME clears HSI_STATUS
                                 47
0014 A0040C            R         48           LD      BX, HSI_TIME            ; Load the HSI_TIME
                                 49
0017 390A09            R         50           JBS     AL, 1, hsi_hi           ; Jump if HSI.0 is high
                                 51
001A C0080C            R         52   hsi_lo: ST      BX, LO_EDGE
001D 48060800          R         53           SUB     HIGH_TIME, LO_EDGE, HI_EDGE
0021 27E7                        54           BR      wait
                                 55
                                 56
0023 C0060C            R         57   hsi_hi: ST      BX, HI_EDGE
0026 48080602          R         58           SUB     LOW_TIME, HI_EDGE, LO_EDGE
002A 27DE                        59           BR      wait
                                 60
002C                             61           END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

3.11. EXAMPLE-4 SCANNING THE A/D CHANNELS

```
MCS-96 MACRO ASSEMBLER    SCANNING THE A TO D CHANNELS

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:ATODX.SRC
OBJECT FILE: :F1:ATODX.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                  1   $TITLE('SCANNING THE A TO D CHANNELS')
                                  2   $PAGELENGTH(95)
                                  3
                                  4   ;       This program scans A to D lines 0 through 3 and stores the
                                  5   ;       results in RESULT_N
                                  6
     0002                         7   AD_RESULT_LO    equ     02
     0003                         8   AD_RESULT_HI    equ     03
     0002                         9   AD_COMMAND      equ     02
     0018                        10   SP              equ     18H
                                 11
                                 12
   0000                          13   dseg
                                 14
   0000                          15   RESULT_TABLE:
   0000                          16           RESULT_1:       dsw     1
   0002                          17           RESULT_2:       dsw     1
   0004                          18           RESULT_3:       dsw     1
   0006                          19           RESULT_4:       dsw     1
                                 20
   0000                          21   rseg
                                 22
   0000                          23           AX:     dsw     1
     0000                        24           AL      equ     AX      :byte
     0001                        25           AH      equ     (AX+1)  :byte
                                 26
   0002                          27           BX:     dsw     1
     0002                        28           BL      equ     BX      :byte
     0003                        29           BU      equ     (BX+1)  :byte   ; Note that 'BH' is an opcode so it
                                 30                                           ; can't be used as a label
                                 31
                                 32
   0004                          33           DX:     dsw     1
     0004                        34           DL      equ     DX      :byte
     0005                        35           DH      equ     (DX+1)  :byte
                                 36
                                 37
   0000                          38   cseg
                                 39
                                 40
   0000 A1C00018                 41   start:  LD      SP, #0C0H       ; Set Stack Pointer
   0004 A00002          R        42           LD      BX, 00H         ; Use the zero register
                                 43
   0007 910802          R        44   next:   ORB     BL, #1000B      ; Start conversion on channel
   000A B00202          R        45           LDB     AD_COMMAND, BL  ; indicated by BL register
   000D 710702          R        46           ANDB    BL, #0111B
                                 47
   0010 FD                       48           NOP             ; Wait for conversion to start
                                 49
   0011 3B02FD                   50   check:  JBS     AD_RESULT_LO, 3, check  ; Wait while A to D is busy
                                 51
                                 52           LDB     AL, AD_RESULT_LO        ; Load low order result
   0014 B00200          R        53           LDB     AH, AD_RESULT_HI        ; Load high order result
   0017 B00301          R        54
   001A 54020204        R        55           ADDB    DL, BL, BL              ; DL=BL*2
   001E AC0404          R        56           LDBZE   DX, DL
   0021 C304000000      R        57           ST      AX, RESULT_TABLE[DX]    ; Store result indexed by BL*2
                                 58
   0026 1702            R        59           INCB    BL
   0028 710302          R        60           ANDB    BL, #03H
                                 61
   002B 27DA                     62           BR      next
                                 63
   002D                          64           END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

3.12. EXAMPLE-5 TABLE LOOKUP-AND INTERPOLATION

```
MCS-96 MACRO ASSEMBLER    TABLE LOOKUP AND INTERPOLATION

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:INTERX.SRC
OBJECT FILE: :F1:INTERX.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                  LINE      SOURCE STATEMENT
                                    1  $TITLE('TABLE LOOKUP AND INTERPOLATION')
                                    2  $PAGELENGTH(95)
                                    3
                                    4  ;       This program uses a lookup table to generate 12-bit function values
                                    5  ;       using 8-bit input values.  The table is 16 bytes long and 16 bits wide.
                                    6  ;       A linear interpolation is made using the following fomula:
                                    7
                                    8  ;              Table_Step        IN_VAL - TABLE_LOW
                                    9  ;       ---------------------- = ------------------ =
                                   10  ;       TABLE_HIGH - TABLE_LOW    OUT - TABLE_LOW
                                   11  ;
                                   12  ;          16      IN_DIF
                                   13  ;       ------- = -------
                                   14  ;       TAB_DIF   OUT_DIF
                                   15
                                   16  ;       Cross Multiplication is used to solve for OUT_DIF
                                   17
                                   18
                                   19
     0018                          20  SP      equ     18H
                                   21
                                   22
     0000                          23  dseg
                                   24
     0000                          25  RESULT_TABLE:
     0000                          26          RESULT:         dsw     1
                                   27
                                   28
     0000                          29  rseg
                                   30
     0000                          31          AX:     dsw     1
       0000                        32          AL      equ     AX      :byte
       0001                        33          AH      equ     (AX+1)  :byte
                                   34
     0002                          35          BX:     dsw     1
       0002                        36          BL      equ     BX      :byte
       0003                        37          BU      equ     (BX+1)  :byte   ; Note that 'BH' is an opcode so it
                                   38                                          ; can't be used as a label
                                   39
     0004                          40          IN_VAL:         dsb     1
     0006                          41          TABLE_LOW:      dsw     1
     0008                          42          TABLE_HIGH:     dsw     1
     000A                          43          IN_DIF:         dsw     1
       000A                        44          IN_DIFB         equ     IN_DIF  :byte
     000C                          45          TAB_DIF:        dsw     1
     000E                          46          OUT:            dsw     1
     0010                          47          OUT_DIF:        dsl     1
                                   48
                                   49
     0000                          50  cseg
                                   51
                                   52
     0000 A1C00018                 53  start:  LD      SP, #0C0H       ; Set Stack Pointer
                                   54
                                   55
     0004 B00400          R        56  look:   LDB     AL, IN_VAL
     0007 180300          R        57          SHRB    AL, #3          ; Place 2 times the upper nibble in byte
     000A 71FE00          R        58          ANDB    AL, #11111110B  ; Insure AL is a word address
     000D AC0000          R        59          LDBZE   AX, AL
                                   60
     0010 A300420006      R        61          LD      TABLE_LOW, TABLE [AX]   ; TABLE_LOW is table output value
                                   62                                          ; of IN_VAL rounded down to the
                                   63                                          ; nearest multiple of 10H.
                                   64
     0015 A300440008      R        65          LD      TABLE_HIGH, (TABLE+2) [AX]  ; TABLE_HIGH is the table output
                                   66                                          ; value of IN_VAL rounded up to the
                                   67                                          ; nearest multiple of 10H.
                                   68
                                   69
     001A 4806080C        R        70          SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW
                                   71
     001E 510F040A        R        72          ANDB    IN_DIFB, IN_VAL, #0FH
     0022 BC0A0A          R        73          LDBSE   IN_DIF, IN_DIFB         ; Make input difference into a word
                                   74
     0025 FE4C0C0A10      R        75          MUL     OUT_DIF, IN_DIF, TAB_DIF
     002A FE8D100010      R        76          DIV     OUT_DIF, #16
                                   77
     002F 4406100E        R        78  labl:   ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                   79                                          ; generated with truncated IN_VAL
                                   80                                          ; as input
     0033 08040E          R        81          SHR     OUT, #4                 ; Round to 12-bit answer
```

```
MCS-96 MACRO ASSEMBLER    TABLE LOOKUP AND INTERPOLATION                10/11/83

ERR LOC   OBJECT                LINE        SOURCE STATEMENT
    0036 D307                    82           JNC     lab2
    0038 070E                R   83           INC     OUT                    ; Round up if Carry = 1
    003A C30100000E          R   84           ST      OUT, RESULT
                                 85
    003F 27C3                    86   lab2:   BR      look
                                 87
                                 88
    0041                         89   cseg
                                 90
    0042 000000200034004C        91   table:  DCW     0000H, 2000H, 3400H, 4C00H     ; A random non-monotonic
    004A 005D006A00720078        92           DCW     5D00H, 6A00H, 7200H, 7800H     ; function
    0052 007B007D0076006D        93           DCW     7B00H, 7D00H, 7600H, 6D00H
    005A 005D004B00340022        94           DCW     5D00H, 4B00H, 3400H, 2200H
    0062 0010                    95           DCW     1000H
                                 96
    0064                         97           END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

3.13. DETAILED INSTRUCTION SET DESCRIPTION

This section gives a description of each instruction recognized by the 8096 sorted alphabetically by the mnemonic used in the assembly language for the 8096. Note that the effect on the program counter (PC) is not always shown in the instruction descriptions. All instructions increment the PC by the number of bytes in the instruction. Several acronynms are used in the instruction set descriptions which are defined here:

aa. A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

aa	Addressing mode
00	Register direct
01	Immediate
10	Indirect
11	Indexed

The selection between indirect and indirect with auto-increment or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to take part in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

breg. A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D."

baop. A byte operand which is addressed by any of the address mooes discussed in section 3.2.

bitno. A three bit field within an instruction op-code which selects one of the eight bits in a byte.

wreg. A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D."

waop. A word operand which is addressed by any of the address modes discussed in section 3.2.

Ireg. A 32-bit register in the internal register file.

BEA. Extra bytes of code required for the address mode selected.

CEA. Extra state times (cycles) required for the address mode selected.

cadd An address in the program code.

Flag Settings. The modification to the flag setting is shown for each instruction. A checkmark (√) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction.

Generic Jumps and Calls. The assembler for the 8096 provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user guide (see section 3.0) should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.

3.13.1. ADD (Two Operands) — ADD WORDS

Operation: The sum of the two word operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

```
      DST    SRC
ADD   wreg,  waop
```

Object Code Format: [011001aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.2. ADD (Three Operands) — ADD WORDS

Operation: The sum of the second and third word operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

```
      DST    SRC1   SRC2
ADD   Dwreg, Swreg, waop
```

Object Code Format: [010001aa][waop][Swreg][Dwreg]

Bytes: 3+BEA
States: 5+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.3. ADDB (Two Operands) — ADD BYTES

Operation: The sum of the two byte operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

```
        DST     SRC
ADDB    breg,   boap
```

Object Code Format: [011101aa][baop][breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.4. ADDB (Three Operands) — ADD BYTES

Operation: The sum of the second and third byte operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
ADDB    Dbreg,  Sbreg,  baop
```

Object Code Format: [010101aa][baop][Sbreg][Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.5. ADDC — ADD WORDS WITH CARRY

Operation: The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

```
        DST     SRC
ADDC    wreg,   waop
```

Object Code Format: [101001aa][waop][wreg]

Bytes: 2 + BEA
States: 4 + BEA

Flags Affected

Z	N	C	V	VT	ST
↓	√	√	√	↑	–

3.13.6. ADDCB — ADD BYTES WITH CARRY

Operation: The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

```
        DST     SRC
ADDCB   breg,   baop
```

Object Code Format: [101101aa][baop][breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
↓	√	√	√	↑	–

3.13.7. AND (Two Operands) — LOGICAL AND WORDS

Operation: The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
        DST     SRC
AND     wreg,   waop
```

Object Code Format: [011000aa][waop][wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.8. AND (Three Operands) — LOGICAL AND WORDS

Operation: The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
AND     Dwreg,  Swreg,  waop
```

Object Code Format: [010000aa][waop][Swreg][Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.9. ANDB (Two Operands) — LOGICAL AND BYTES

Operation: The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
        DST     SRC
ANDB    breg,   baop
```

Object Code Format: [011100aa][baop][breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.10. ANDB (Three Operands) — LOGICAL AND BYTES

Operation: The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
ANDB    Dbreg,  Sbreg,  baop
```

Object Code Format: [010100aa][baop][Sbreg][Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.11. BR (Indirect) — BRANCH INDIRECT

Operation: The execution continues at the address specified in the operand word register.

PC ← (DEST)

Assembly Language Format: BR [wreg]

Object Code Format: [11100011][wreg]

Bytes: 2
States: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.12. CLR — CLEAR WORD

Operation: The value of the word operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLR wreg

Object Code Format: [00000001][wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
1	0	0	0	–	–

3.13.13. CLRB — CLEAR BYTE

Operation: The value of the byte operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLRB breg

Object Code Format: [00010001][breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
1	0	0	0	–	–

3.13.14. CLRC — CLEAR CARRY FLAG

Operation: The value of the carry flag is set to zero.

C ← 0

Assembly Language Format: CLRC

Object Code Format: [11111000]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	0	–	–	–

3.13.15. CLRVT — CLEAR OVERFLOW TRAP

Operation: The value of the overflow-trap flag is set to zero.

VT ← 0

Assembly Language Format: CLRVT

Object Code Format: [11111100]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	0	–

3.13.16. CMP — COMPARE WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

	DST	SRC
CMP	wreg,	waop

Object Code Format: [100010aa][waop][wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.17. CMPB — COMPARE BYTES

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

	DST	SRC
CMPB	breg,	baop

Object Code Format: [100110aa][baop][breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.18. DEC — DECREMENT WORD

Operation: The value of the word operand is decremented by one.

(DEST) ← (DEST) — 1

Assembly Language Format: DEC wreg

Object Code Format: [00000101][wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.19. DECB — DECREMENT BYTE

Operation: The value of the byte operand is decremented by one.

(DEST) ← (DEST) — 1

Assembly Language Format: DECB breg

Object Code Format: [00010101] [breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.20. DI — DISABLE INTERRUPTS

Operation: Interrupts are disabled. Interrupt-calls will not occur after this instruction.

Interrupt Enable (PSW.9) ← 0

Assembly Language Format: DI

Object Code Format: [11111010]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.21. DIV — DIVIDE INTEGERS

Operation: This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
      DST    SRC
DIV   lreg,  waop
```

Object Code Format: [11111110][100011aa][waop][lreq]

Bytes: 2+BEA
States: 29 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	√	↑	–

3.13.22. DIVB — DIVIDE SHORT-INTEGERS

Operation: This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e. the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
      DST    SRC
DIVB  wreg,  baop
```

Object Code Format: [11111110][100111aa][baop][wreg]

Bytes: 2+BEA
States: 21 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	√	↑	–

3.13.23. DIVU — DIVIDE WORDS

Operation: This instruction divides the contents of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order byte will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVU    lreg,   waop
```

Object Code Format: [100011aa][waop][lreq]

Bytes: 2+BEA
States: 25 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	√	↑	–

3.13.24. DIVUB — DIVIDE BYTES

Operation: This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVUB   wreg,   baop
```

Object Code Format: [100111aa][baop][wreg]

Bytes: 2+BEA
States: 17 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	√	↑	–

3.13.25. DJNZ — DECREMENT AND JUMP IF NOT ZERO

Operation: The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

```
(COUNT) ← (COUNT) — 1
if (COUNT) < > 0 then
  PC ← PC + disp (sign-extended to 16 bits)
end_if
```

Assembly Language Format: DJNZ breg,cadd

Object Code Format: [11100000][breg][disp]

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.26. EI — ENABLE INTERRUPTS

Operation: Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

```
Interrupt Enable (PSW.9) ← 1
```

Assembly Language Format: EI

Object Code Format: [11111011]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.27. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER

Operation: The low order word of the operand is sign-extended throughout the high order word of the operand.

```
if (low word DEST)<8000H then
   (high word DEST) ← 0
else
   (high word DEST) ← 0FFFFH
end_if
```

Assembly Language Format: EXT lreg

Object Code Format: [00000110][lreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.28. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER

Operation: The low order byte of the operand is sign-extended throughout the high order byte of the operand.

```
if (low byte DEST)<80H then
   (high byte DEST) ← 0
else
   (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format: EXTB wreg

Object Code Format: [00010110][wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.29. INC — INCREMENT WORD

Operation: The value of the word operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INC wreg

Object Code Format: [00000111] [wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.30. INCB — INCREMENT BYTE

Operation: The value of the byte operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INCB breg

Object Code Format: [00010111] [breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.31. JBC — JUMP IF BIT CLEAR

Operation: The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is set (i.e., 1), control passes to the next sequential instruction.

```
if (specified bit) = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JBC breg,bitno,cadd

Object Code Format: [00110bbb][breg][disp]

where bbb is the bit number within the specified register.

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.32. JBS — JUMP IF BIT SET

Operation: The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

```
if (specified bit) = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JBS breg,bitno,cadd

Object Code Format: [00111bbb][breg][disp]

where bbb is the bit number within the specified register.

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.33. JC — JUMP IF CARRY FLAG IS SET

Operation: If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if C = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JC cadd

Object Code Format: [11011011][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.34. JE — JUMP IF EQUAL

Operation: If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JE cadd

Object Code Format: [11011111][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.35. JGE — JUMP IF SIGNED GREATER THAN OR EQUAL

Operation: If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

```
if N = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGE cadd

Object Code Format: [11010110][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.36. JGT — JUMP IF SIGNED GREATER THAN

Operation: If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

```
if N = 0 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGT cadd

Object Code Format: [11010010][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.37. JH — JUMP IF HIGHER (UNSIGNED)

Operation: If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

```
if C = 1 and Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JH cadd

Object Code Format: [11011001][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.38. JLE — JUMP IF SIGNED LESS THAN OR EQUAL

Operation: If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If both the negative flag and the zero flag are clear (i.e, 0), control passes to the next sequential instruction.

```
if N = 1 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLE cadd

Object Code Format: [11011010][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.39. JLT — JUMP IF SIGNED LESS THAN

Operation: If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLT cadd

Object Code Format: [11011110][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.40. JNC — JUMP IF CARRY FLAG IS CLEAR

Operation: If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

```
if C = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNC cadd

Object Code Format: [11010011] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.41. JNE — JUMP IF NOT EQUAL

Operation: If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

```
if Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNE cadd

Object Code Format: [11010111] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.42. JNH — JUMP IF NOT HIGHER (UNSIGNED)

Operation: If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is set (i.e., 1), or the zero flag is not, control passes to the next sequential instruction.

```
if C = 0 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNH cadd

Object Code Format: [11010001][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.43. JNST — JUMP IF STICKY BIT IS CLEAR

Operation: If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

```
if ST = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNST cadd

Object Code Format: [11010000][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.44. JNV — JUMP IF OVERFLOW FLAG IS CLEAR

Operation: If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

```
if V = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNV cadd

Object Code Format: [11010101][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.45. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR

Operation: If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction.

```
if VT = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNVT cadd

Object Code Format: [11010100][disp]

Bytes: 2
States: Jump Not Taken: 4
Jumps Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	0	–

3.13.46. JST — JUMP IF STICKY BIT IS SET

Operation: If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if ST = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JST cadd

Object Code Format: [11011000][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.47. JV — JUMP IF OVERFLOW FLAG IS SET

Operation: If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if V = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JV cadd

Object Code Format: [11011101][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.48. JVT — JUMP IF OVERFLOW TRAP IS SET

Operation: If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if VT = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JVT cadd

Object Code Format: [11011100][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	0	–

3.13.49. LCALL — LONG CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

```
SP ← SP – 2
(SP) ← PC
PC ← PC + disp
```

Assembly Language Format: LCALL cadd

Object Code Format: [11101111][disp-low][disp-hi]

Bytes: 3
States: Onchip stack: 13
Onchip stack: 16

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.50. LD — LOAD WORD

Operation: The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
      DST     SRC
LD    wreg,   waop
```

Object Code Format: [101000aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.51. LDB — LOAD BYTE

Operation: The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
      DST     SRC
LDB   breg,   baop
```

Object Code Format: [101100aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.52. LDBSE — LOAD INTEGER WITH SHORT-INTEGER

Operation: The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
if (SRC) < 80H then
   (high byte DEST) ← 0
else
   (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format:

```
          DST     SRC
LDBSE     wreg,   baop
```

Object Code Format: [101111aa][baop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.53. LDBZE — LOAD WORD WITH BYTE

Operation: The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
(high byte DEST) ← 0
```

Assembly Language Format:

```
          DST     SRC
LDBZE     wreg,   baop
```

Object Code Format: [101011aa][baop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.54. LJMP — LONG JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

PC ← PC + disp

Assembly Language Format: LJMP cadd

Object Code Format: [11100111][disp-low][disp-hi]

Bytes: 3
States: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.55. MUL (Two Operands) — MULTIPLY INTEGERS

Operation: The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MUL     lreg,   waop
```

Object Code Format: [11111110][011011aa][waop][lreg]

Bytes: 3+BEA
States: 29 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	✓

3.13.56. MUL (Three Operands) — MULTIPLY INTEGERS

Operation: The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
          DST     SRC1    SRC2
MUL       lreg,   wreg,   waop
```

Object Code Format: [11111110][010011aa][waop][wreg][lreg]

Bytes: 4 + BEA
States: 30 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.57. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS

Operation: The two SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
          DST     SRC
MULB      wreg,   baop
```

Object Code Format: [11111110][011111aa][baop][wreg]

Bytes: 3 + BEA
States: 21 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.58. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS

Operation: The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
          DST     SRC1    SRC2
MULB      wreg,   breg    baop
```

Object Code Format: [11111110][010111aa][baop][breg][wreg]

Bytes: 4 + BEA
States: 22 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.59. MULU (Two Operands) — MULTIPLY WORDS

Operation: The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
          DST     SRC
MULU      lreg,   waop
```

Object Code Format: [011011aa][waop][lreg]

Bytes: 2 + BEA
States: 25 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.60. MULU (Three Operands) — MULTIPLY WORDS

Operation: The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST)< — (SRC1) * (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
MULU    lreg,   wreg,   waop
```

Object Code Format: [010011aa][waop][wreg][lreg]

Bytes: 3+BEA
States: 26 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	✓

3.13.61. MULUB (Two Operands) — MULTIPLY BYTES

Operation: The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MULUB   wreg,   baop
```

Object Code Format: [011111aa][baop][wreg]

Bytes: 2+BEA
States: 17 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	✓

3.13.62. MULUB (Three Operands) — MULTIPLY BYTES

Operation: The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
          DST     SRC1    SRC2
MULUB     wreg,   breg,   baop
```

Object Code Format: [010111aa][baop][breg][wreg]

Bytes: 3+BEA
States: 18 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.63. NEG — NEGATE INTEGER

Operation: The value of the INTEGER operand is negated.

(DEST) ← – (DEST)

Assembly Language Format: NEG wreg

Object Code Format: [00000011][wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.64. NEGB — NEGATE SHORT-INTEGER

Operation: The value of the SHORT-INTEGER operand is negated.

(DEST) ← −(DEST)

Assembly Language Format: NEGB breg

Object Code Format: [00010011][breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.65. NOP — NO OPERATION

Operation: Nothing is done. Control passes to the next sequential instruction.

Assembly Language Format: NOP

Object Code Format: [11111101]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.66. NORML — NORMALIZE LONG-INTEGER

Operation: The LONG-INTEGER operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

```
(COUNT) ← 0
do while (MSB(DEST) = 0) AND ((COUNT) < 31)
        (DEST) ← (DEST) * 2
        (COUNT) ← (COUNT) + 1
end_while
```

Assembly Language Format: NORML lreg,breg

Object Code Format: [00001111][breg][lreg]

Bytes: 3
States: 8 + No. of shifts performed

Flags Affected

Z	N	C	V	VT	ST
√	1	0	–	–	–

3.13.67. NOT — COMPLEMENT WORD

Operation: The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

```
(DEST) ← NOT(DEST)
```

Assembly Language Format: NOT wreg

Object Code Format: [00000010][wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.68. NOTB — COMPLEMENT BYTE

Operation: The value of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

Assembly Language Format: NOTB breg

Object Code Format: [00010010][breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.69. OR — LOGICAL OR WORDS

Operation: The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format:

	DST	SRC
OR	wreg,	waop

Object Code Format: [100000aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.70. ORB — LOGICAL OR BYTES

Operation: The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format: ORB breg,baop

Object Code Format: [100100aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.71. POP — POP WORD

Operation: The word on top of the stack is popped and placed at the destination operand.

(DEST) ← (SP)
SP ← SP + 2

Assembly Language Format: POP waop

Object Code Format: [110011aa][waop]

Bytes: 1+BEA
States: Onchip Stack: 12+CEA
Offchip Stack 14+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.72. POPF — POP FLAGS

Operation: The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

(PSW) ← (SP)
SP ← SP + 2

Assembly Language Format: POPF

Object Code Format: [11110011]

Bytes: 1
States: Onchip Stack: 9
Offchip Stack: 13

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	√	√

3.13.73. PUSH — PUSH WORD

Operation: The specified operand is pushed onto the stack.

SP ← SP − 2
(SP) ← (DEST)

Assembly Language Format: PUSH waop

Object Code Format: [110010aa] [waop]

Bytes: 1 + BEA
States: Onchip Stack: 8 + CEA
Offchip Stack: 12 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.74. PUSHF — PUSH FLAGS

Operation: The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

SP ← SP − 2
(SP) ← PSW
PSW ← 0

Assembly Language Format: PUSHF

Object Code Format: [11110010]

Bytes: 1
States: Onchip Stack: 8
Offchip Stack: 12

Flags Affected

Z	N	C	V	VT	ST
0	0	0	0	0	0

3.13.75. RET — RETURN FROM SUBROUTINE

Operation: The PC is popped off the top of the stack.

PC ← (SP)
SP ← SP + 2

Assembly Language Format: RET

Object Code Format: [11110000]

Bytes: 1
States: Onchip Stack: 12
Offchip Stack: 16

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.76. RST — RESET SYSTEM

Operation: The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value (see section 2.15.2, "Reset Status"). Executing this instruction will cause a pulse to appear on the reset pin of the 8096.

PSW ← 0
PC ← 2080H

Assembly Language Format: RST

Object Code Format: [11111111]

Bytes: 1
States: 16

Flags Affected

Z	N	C	V	VT	ST
0	0	0	0	0	0

3.13.77. SCALL — SHORT CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

SP ← SP — 2
(SP) ← PC
PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SCALL cadd

Object Code Format: [00101xxx][disp-low]

where xxx holds the three high-order bits of displacement.

Bytes: 2
States: Onchip Stack: 13
Offchip Stack: 16

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.78. SETC — SET CARRY FLAG

Operation: The carry flag is set.

C ← 1

Assembly Language Format: SETC

Object Code Format: [11111001]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	1	–	–	–

3.13.79. SHL — SHIFT WORD LEFT

Operation: The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <>0
    C ← High order bit of (DEST)
    (DEST) ← (DEST)*2
    Temp ← Temp — 1
end_while
```

Assembly Language Format:

```
    SHL   wreg,#count
or
    SHL   wreg,breg
```

Object Code Format: [00001001] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.80. SHLB — SHIFT BYTE LEFT

Operation: The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
     C ← High order bit of (DEST)
     (DEST) ← (DEST)*2
     Temp ← Temp  –  1
end_while
```

Assembly Language Format:

```
     SHLB   breg,#count
or
     SHLB   breg,breg
```

Object Code Format: [00011001][cnt/breg][breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.81. SHLL — SHIFT DOUBLE-WORD LEFT

Operation: The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
     C ← High order bit of (DEST)
     (DEST) ← (DEST)*2
     Temp ← Temp - 1
end_while
```

Assembly Language Format:

```
     SHLL   lreg,#count
or
     SHLL   lreg,breg
```

Object Code Format: [00001101][cnt/breg][lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.82. SHR — LOGICAL RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp - 1
end_while
```

Assembly Language Format:

```
    SHR   wreg,#count
or
    SHR   wreg,breg
```

Object Code Format: [00001000][cnt/breg][wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.83. SHRA — ARITHMETIC RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp - 1
end_while
```

Assembly Language Format:

```
    SHRA   wreg,#count
or
    SHRA   wreg,breg
```

Object Code Format: [00001010][cnt/breg][wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.84. SHRAB — ARITHMETIC RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C, = Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where /is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
    SHRAB   breg,#count
or
    SHRAB   breg,breg
```

Object Code Format: [00011010] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.85. SHRAL — ARITHMETIC RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where/is signed division
    Temp ← Temp – 1
end_while
```

Assembly Language Format:

```
    SHRAL   lreg,#count
or
    SHRAL   lreg,breg
```

Object Code Format: [00001110][cnt/breg][lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.86. SHRB — LOGICAL RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where/is unsigned division
    Temp ← Temp  − 1
end_while
```

Assembly Language Format:

```
    SHRB   breg,#count
or
    SHRB   breg,breg
```

Object Code Format: [00011000] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.87. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
     C ← Low order bit of (DEST)
     (DEST) ← (DEST) / 2 where/is unsigned division
     Temp ← Temp  −  1
end_while
```

Assembly Language Format:

```
     SHRL   lreg,#count
or
     SHRL   lreg,breg
```

Object Code Format: [00001100] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.88. SJMP — SHORT JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SJMP cadd

Object Code Format: [00100xxx][disp-low]

where xxx holds the three high order bits of the displacement.

Bytes: 2
States: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.89. SKIP — TWO BYTE NO-OPERATION

Operation: Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

Assembly Language Format: SKIP breg

Object Code Format: [00000000][breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.90. ST — STORE WORD

Operation: The value of the leftmost word operand is stored into the rightmost operand.

(DEST) ← (SRC)

Assembly Language Format:

	SRC	DST
ST	wreg,	waop

Object Code Format: [110000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.91. STB — STORE BYTE

Operation: The value of the leftmost byte operand is stored into the rightmost operand.

(DEST) ← (SRC)

Assembly Language Format:

	SRC	DST
STB	breg,	baop

Object Code Format: [110001aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.92. SUB (Two Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUB     wreg,   waop
```

Object Code Format: [011010aa][waop][wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.93. SUB (Three Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2,
SUB     wreg,   wreg,   waop
```

Object Code Format: [010010aa][waop][Swreg][Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.94. SUBB (Two Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUBB    breg,   baop
```

Object Code Format: [011110aa][baop][breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.95. SUBB (Three Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUBB    breg,   baop
```

Object Code Format: [010110aa][baop][Sbreg][Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.96. SUBC — SUBTRACT WORDS WITH BORROW

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

```
        DST     SRC
SUBC    wreg,   waop
```

Object Code Format: [101010aa][waop][wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	–

3.13.97. SUBCB — SUBTRACT BYTES WITH BORROW

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

```
        DST     SRC
SUBCB   breg,   baop
```

Object Code Format: [101110aa][baop][breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	–

3.13.98. TRAP — SOFTWARE TRAP

Operation: This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

SP ← SP − 2
(SP) ← PC
PC ← (2010H)

Assembly Language Format: This instruction is not supported by revision 1.0 of the 8096 assembly language.

Object Code Format: [11110111]

Bytes: 1
States: Onchip Stack: 21
Offchip Stack: 24

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.99. XOR — LOGICAL EXCLUSIVE-OR WORDS

Operation: The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

```
        DST     SRC
XOR     wreg,   waop
```

Object Code Format: [100001aa][waop][wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.100. XORB — LOGICAL EXCLUSIVE-OR BYTES

Operation: The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

	DST	SRC
XORB	breg,	baop

Object Code Format: [100101aa][baop][breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

Hardware Design Information 4

4 Hardware Design Information

CHAPTER 4
HARDWARE DESIGN INFORMATION

4.0. HARDWARE INTERFACING OVERVIEW

This section of the manual is devoted to the hardware engineer. All of the information you need to connect the correct pin to the correct external circuit is provided. Many of the special function pins have different characteristics which are under software control, therefore, it is necessary to define the system completely before the hardware is wired-up.

Frequently within this section a specification for a current, voltage, or time period is referred to; the values provided are to be used as an approximation only. The exact specification can be found in the latest data sheet for the particular part and temperature range that is being used.

4.1. REQUIRED HARDWARE CONNECTIONS

Although the 8096 is a single-chip microcontroller, it still requires several external connections to make it work. Power must be applied, a clock source provided, and some form of reset circuitry must be present. We will look at each of these areas of circuitry separately. Figure 4-5 shows the connections that are needed for a single-chip system.

4.1.1. Power Supply Information

Power for 8096 flows through 6 pins; one VCC pin, two VSS pins, one VREF (analog VCC), one ANGND (Analog VSS), and one VPD (V Power Down) pin. All six of these pins must be connected to the 8096 for normal operation. The VCC pin, VREF pin and VPD pin should be tied to 5 volts. When the analog to digital converter is being used it may be desirable to connect the VREF pin to a separate power supply, or at least a separate power supply line.

The two VSS pins should be connected together with as short a lead as possible to avoid problems due to voltage drops across the wiring. There should be no measurable voltage difference between VSS1 and VSS2. The 2 VSS pins and the ANGND pin should all be nominally at 0 volts. The maximum current drain of the 8096 is around 200mA, with all lines unloaded.

When the analog converter is being used, clean, stable power must be provided to the analog section of the chip to assure highest accuracy. To achieve this, it may be desirable to separate the analog power supply from the digital power supply. The VREF pin supplies 5 volts to the analog circuitry and the ANGND pin is the ground for this section of the chip. More information on the analog power supply is in section 4.3.1.

4.1.2. Other Needed Connections

Several of the pins on the 8096 are used to configure the mode of operation. In normal operation the following pins should be tied directly to the indicated power supply.

PIN	POWER SUPPLY
NMI	VCC
$\overline{\text{TEST}}$	VCC
$\overline{\text{EA}}$	VCC (to allow internal execution) VSS (to force external execution)

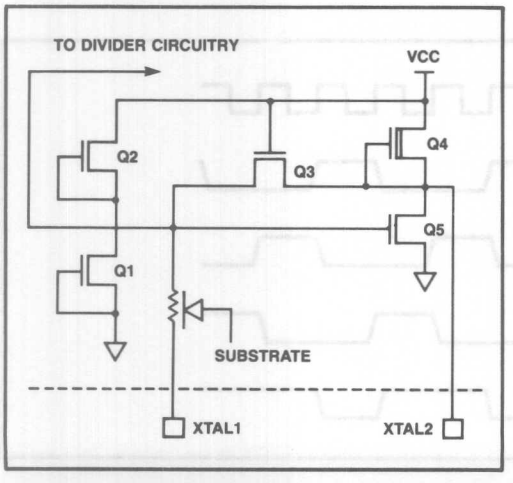


Figure 4-1. 8096 Oscillator Circuit

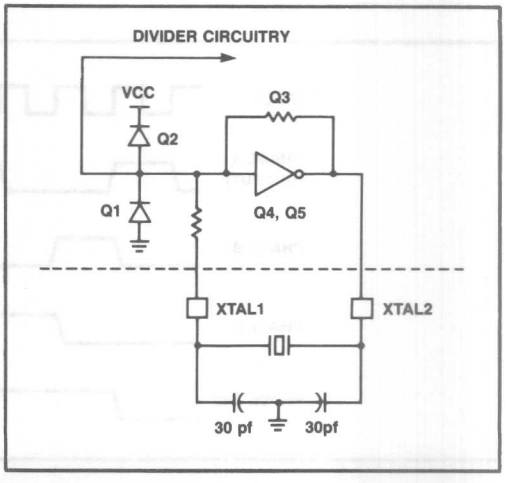


Figure 4-2. Crystal Oscillator Circuit

Although the $\overline{EA}$ pin has an internal pulldown, it is best to tie this pin to the desired level if it is not left completely disconnected. This will prevent induced noise from disturbing the system.

4.1.3. Oscillator Information

The 8096 requires a clock source to operate. This clock can be provided to the chip through the XTAL1 input or the on-chip oscillator can be used. The frequency of operation is from 6.0 MHz to 12 MHz.

The on-chip circuitry for the 8096 oscillator is a single stage linear inverter as shown in Figure 4-1. It is intended for use as a crystal-controlled, positive reactance oscillator with external connections as shown in Figure 4-2. In this application, the crystal is being operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

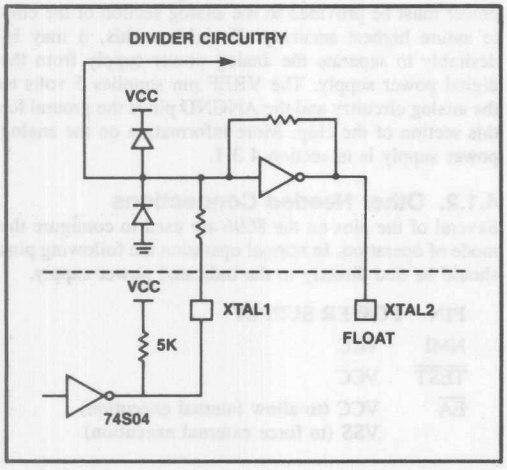


Figure 4-3. External Clock Drive

The crystal specifications and capacitance values (C1 and C2 in Figure 4-2) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. For 0.5% frequency accuracy, the crystal frequency can be specified at series resonance or for parallel resonance with any load capacitance. (In other words, for that degree of frequency accuracy, the load capacitance simply doesn't matter.) For 0.05% frequency accuracy the crystal frequency should be specified for parallel resonance with 25 pF load capacitance, if C1 and C2 are 30 pF.

A more in-depth discussion of crystal specifications and the selection of values for C1 and C2 can be found in the Intel Application Note, AP-155, "Oscillators for Microcontrollers."

To drive the 8096 with an external clock source, apply the external clock signal to XTAL1 and let XTAL2 float. An example of this circuit is shown in Figure 4-3. The required voltage levels on XTAL1 are specified in the data sheet. The signal on XTAL1 must be clean with good solid levels. It is important that the minimum high and low times are met.

There is no specification on rise and fall times, but they should be reasonably fast (on the order of 30 nanoseconds) to avoid having the XTAL1 pin in the transition range for long periods of time. The longer the signal is in the transition region, the higher the probability that an external noise glitch could be seen by the clock generator circuitry. Noise glitches on the 8096 internal clock lines will cause unreliable operation.

The clock generator provides a 3 phase clock output from the XTAL1 pin input. Figure 4-4 shows the waveforms of the major internal timing signals.

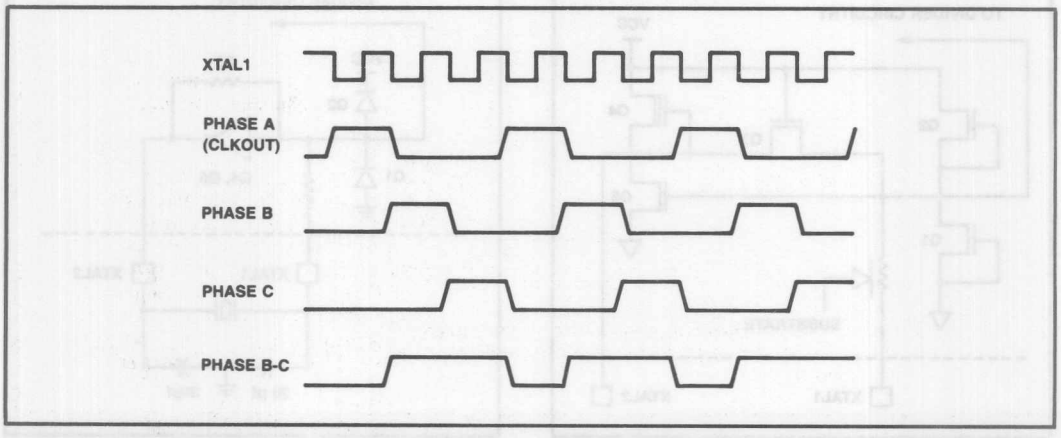


Figure 4-4. Internal Timings

4.1.4. Reset Information

In order for the 8096 to function properly it must be reset. This is done by holding the reset pin low for at least 2 state times after the power supply is within tolerance, the oscillator has stabilized, and the back-bias generator has stabilized. Typically, the back-bias generator requires one millisecond to stabilize.

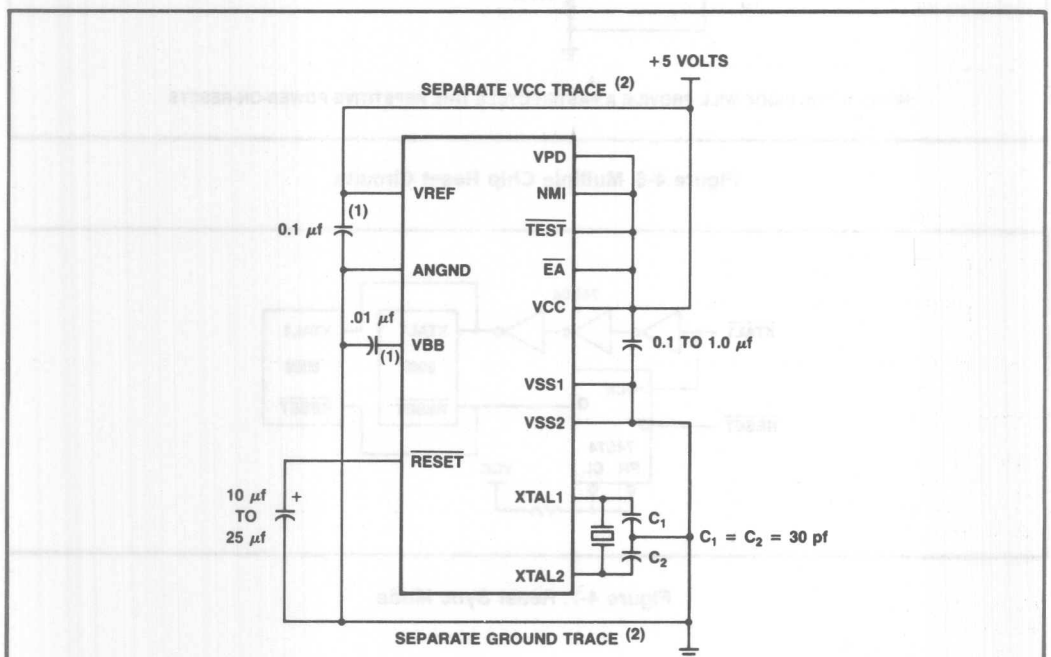
There are several ways of doing this, the simplest being just to connect a capacitor from the reset pin to ground. The capacitor should be on the order of 1 to 2 microfarads for every millisecond of reset time required. This method will only work if the rise time of VCC is fast and the total reset time is less than around 50 milliseconds. It also may not work if the reset pin is to be used to reset other parts on the board. An 8096 with the minimum required connections is shown in Figure 4-5.

The 8096 $\overline{\text{RESET}}$ pin can be used to allow other chips on the board to make use of the watchdog timer or the RST instruction. When this is done the reset hardware should be a one-shot with an open-collector output. The reset pulse going to the other parts may have to be buffered and lengthened with a one-shot, since the $\overline{\text{RESET}}$ low duration is only two state times. If this is done, it is possible that the 8096 will be reset and start running before the other parts on the board are out of reset. The software must account for this possible problem.

A capacitor directly connected to $\overline{\text{RESET}}$ cannot be used to reset the part if the pin is to be used as an output. If a large capacitor is used, the pin will pull down more slowly than normal. It will continue to pull down until the 8096 is reset. It could fall so slowly that it never goes below the internal switch point of the reset signal (1 to 1.5 volts), a voltage which may be above the guaranteed switch point of external circuitry connected to the pin. Several circuit examples are shown in Figure 4-6.

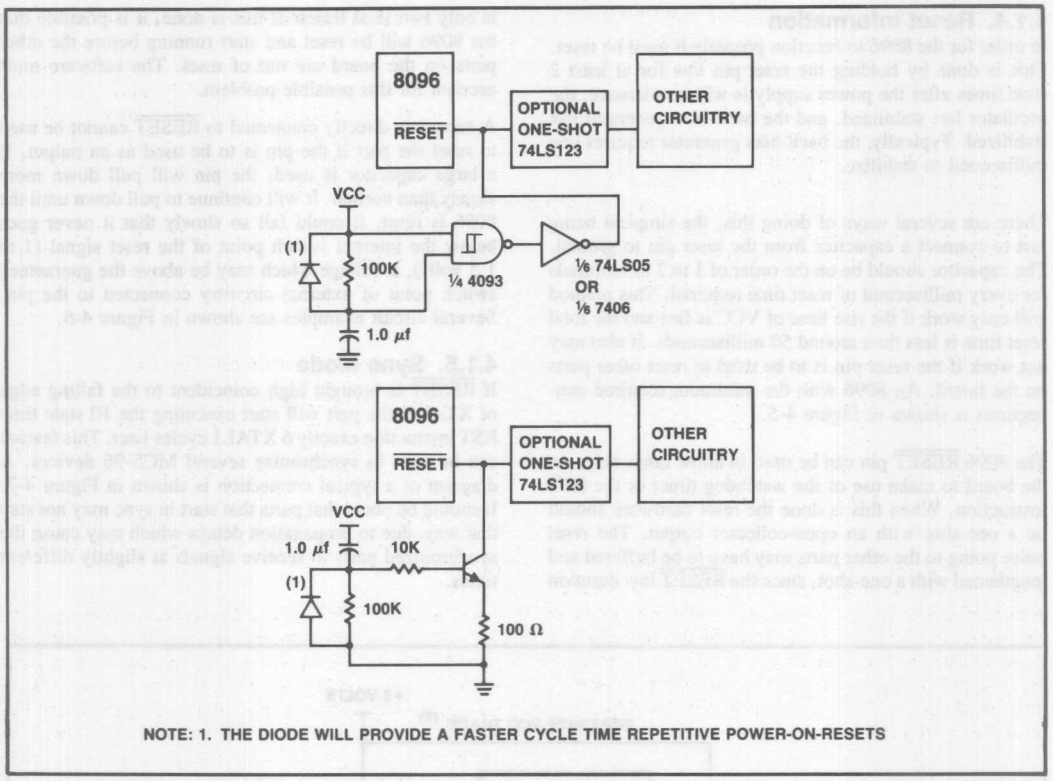
4.1.5. Sync Mode

If $\overline{\text{RESET}}$ is brought high coincident to the falling edge of XTAL1, the part will start executing the 10 state time RST instruction exactly 6 XTAL1 cycles later. This feature can be used to synchronize several MCS-96 devices. A diagram of a typical connection is shown in Figure 4-7. It should be noted that parts that start in sync may not stay that way, due to propagation delays which may cause the synchronized parts to receive signals at slightly different times.



NOTES: 1. THESE CAPACITORS ARE NEEDED ONLY IF A TO D IS USED.
2. VREF & ANGND MAY BE CONNECTED TO THE SAME TRACES AS THE DIGITAL POWER SUPPLY IF THE A TO D IS NOT USED.

Figure 4-5. Minimum Hardware Connections



NOTE: 1. THE DIODE WILL PROVIDE A FASTER CYCLE TIME REPETITIVE POWER-ON-RESETS

Figure 4-6. Multiple Chip Reset Circuits

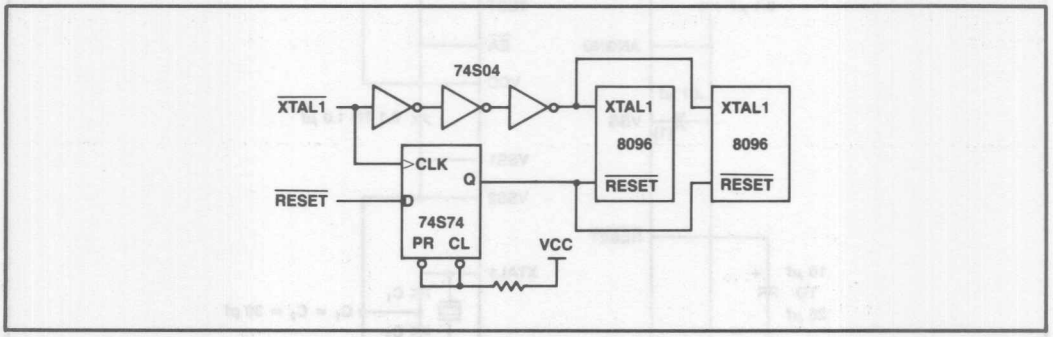


Figure 4-7. Reset Sync Mode

4.1.6. Disabling the Watchdog Timer

The watchdog timer will pull the RESET pin low when it overflows. If the pin is being externally held above the low going threshold, the pull-down transistor will remain on indefinitely. This means that once the watchdog overflows, the part must be reset or RESET must be held high indefinitely. Just resetting the watchdog timer will not reset the flip-flop which keeps the RESET pull-down on.

The pull-down is capable of sinking on the order of 30 milliamps if it is held at 2.0 volts. This amount of current

may cause some long term reliability problems due to localized chip heating. For this reason, parts that will be used in production should never have had the watchdog timer over-ridden for more than a second or two.

Whenever the reset pin is being pulled high while the pull-down is on, it should be through a resistor that will limit the voltage on RESET to 2.5 volts and the current through the pin to 40 milliamps. Figure 4-8 shows a circuit which will provide the desired results. Using the LED will provide the additional benefit of having a visual indicator that the part is trying to reset itself, although this circuit only works at room temperature and VCC = 5 Volts.

4.1.7. Power Down Circuitry

Battery backup can be provided on the 8096 with a 1 mA current drain at 5 volts. This mode will hold locations 0F0H through 0FFH valid as long as the power to the VPD pin remains on. The required timings to put the part into power-down and an overview of this mode are given in section 2.4.2.

A 'key' can be written into power-down RAM while the part is running. This key can be checked on reset to determine if it is a start-up from power-down or a complete cold start. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096 until it has completed its reset operation.

4.2. DRIVE AND INTERFACE LEVELS

There are 5 types of I/O lines on the 8096. Of these, 2 are inputs and 3 are outputs. All of the pins of the same type have the same current/voltage characteristics. Some of the control input pins, such as XTAL1 and RESET, may have slightly different characteristics. These pins are discussed in section 4.1.

While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the latest version of the 8096 Data Sheet.

4.2.1. Quasi-Bidirectional Ports

The quasi-bidirectional port is both an input and an output port. It has three states, low impedance current sink, low impedance current source, and high impedance current source. As a low impedance current sink, the pin has a specification of sinking up to around .4 milliamps, while staying below 0.45 volts. The pin is placed in this condition by writing a '0' to the SFR (Special Function Register) controlling the pin.

When a '1' is written to the SFR location controlling the pin, a low impedance current source is turned on for one state time, then it is turned off and the depletion pull-up holds the line at a logical '1' state. The low-impedance pull-up is used to shorten the rise time of the pin, and has current source capability on the order of 100 times that of the depletion pull-up. The configuration of a quasi-bidirectional port pin is shown in Figure 4-9.

While the depletion mode pull-up is the only device on, the pin may be used as an input with a leakage of around 100 microamps from 0.45 volts to VCC. It is ideal for use with TTL or CMOS chips and may even be used

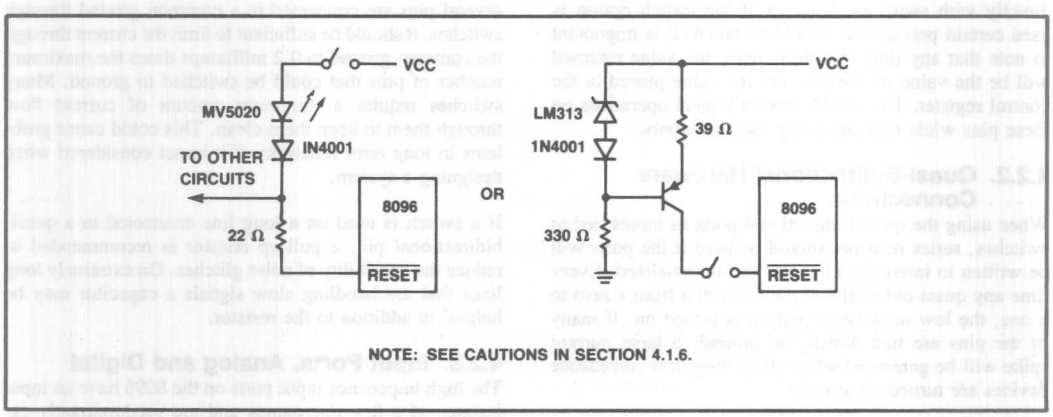
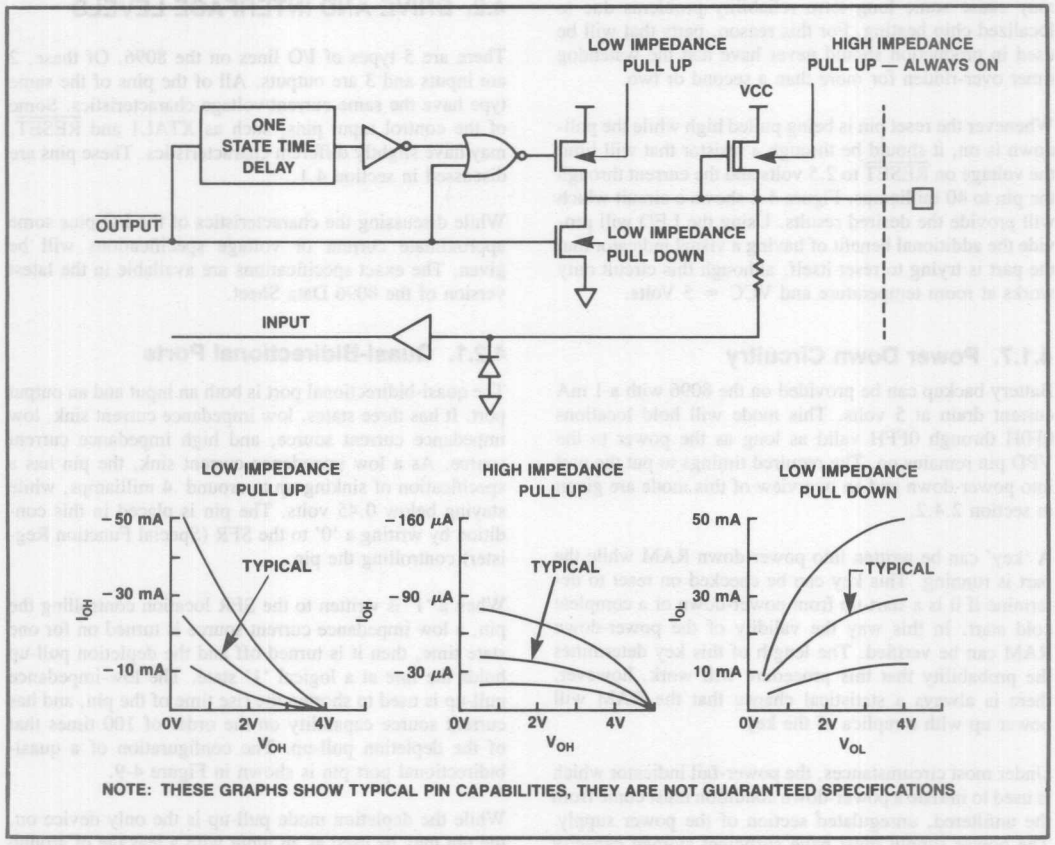


Figure 4-8. Disabling the WDT



NOTE: THESE GRAPHS SHOW TYPICAL PIN CAPABILITIES, THEY ARE NOT GUARANTEED SPECIFICATIONS

Figure 4-9. Quasi-Bidirectional Port

directly with switches, however if the switch option is used certain precautions should be taken. It is impootant to note that any time the pin is read, the value returned will be the value on the pin, not the value placed in the control register. This could prevent logical operations on these pins while they are being used as inputs.

4.2.2. Quasi-Bidirectional Hardware Connections

When using the quasi-bidirectional ports as inputs tied to switches, series resistors should be used if the ports will be written to internally after the part is initialized. Every time any quasi-bidirectional pin is written from a zero to a one, the low impedance pull-up is turned on. If many of the pins are tied directly to ground, a large current spike will be generated when all of these low impedance devices are turned on at once.

For this reason, a series resistor is recommended to limit the current to a maximum of 0.2 milliamps per pin. If several pins are connected to a common ground through switches, it should be sufficient to limit the current through the common ground to 0.2 milliamps times the maximum number of pins that could be switched to ground. Many switches require a minimum amount of current flow through them to keep them clean. This could cause problems in long term reliability if it is not considered when designing a system.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pull-up resistor is recommended to reduce the possibility of noise glitches. On extremely long lines that are handling slow signals a capacitor may be helpful in addition to the resistor.

4.2.3. Input Ports, Analog and Digital

The high impedance input ports on the 8096 have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pf. The Port 0 pins have an additional function when the A to D converter is being

used. These pins are the input to the A to D converter, and as such, are required to provide current to the comparator when a conversion is in process. This means that the input characteristics of a pin will change if a conversion is being done on that pin. See section 4.3.1.

4.2.4. Open Drain Ports

Ports 3 and 4 on the 8096 are open drain ports. There is no pull-up when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to ports 3 and 4 and the Bus pins is shown in Figure 4-10.

When Ports 3 and 4 are to be used as inputs, or as Bus pins, they must first be written with a '1', this will put the ports in a high impedance mode. When they are used as outputs, a pull-up resistor must be used externally. The sink capability of these pins is on the order of 0.4 milliamps so the total pull-up current to the pin must be less than this. A 15k pull-up resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.

4.2.5. HSO Pins, Control Outputs and Bus Pins

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are; TXD, RXD (in mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, and $\overline{WR}$. The bus pins have 3 states; output high, output low, and high impedance input. As a high output, the pins are specified to source around 200 μA to 2.4 volts, but the pins can source on the order of ten times that value in order to provide fast rise times. When used as a low output, the pins can sink around 2 mA at .45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 4-10 shows the internal configuration of a bus pin.

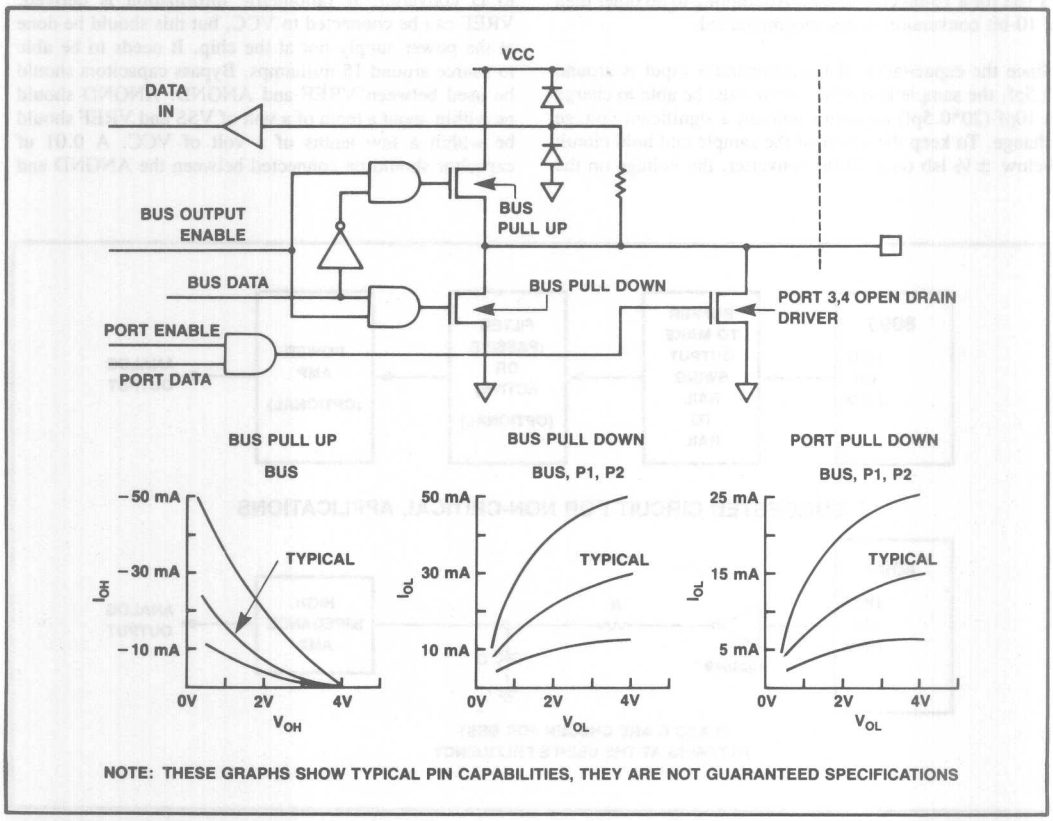


Figure 4-10. Bus and Port 3 and 4 Pins

4.3. ANALOG INTERFACE

Interfacing the 8096 to analog signal can be done in several ways. If the 8096 needs to measure an analog signal the A to D converter can be used. Creation of analog outputs can be done with either the PWM output or the HSO unit.

4.3.1. Analog Inputs

The 8096 can have 8 analog inputs and can convert one input at a time into a digital value. Each conversion takes 42 microseconds with a 12 MHz signal on XTAL1. The input signal is applied to one of the Port 0/Analog Channel inputs. Since there is no sample and hold on the A to D, the input signal must remain constant over the sampling period.

Whan a conversion takes place, the 8096 compares the external signal to that of its internal D to A. Based on the result of the comparison it adjusts the D to A and compares again. Each comparison takes 8 state times and requires the input to the comparator to be charged up. 20 comparisons are made during a conversion, two times for each bit of resolution. An additional 8 states are used to load and store values. The total number of state times required is 168 for a 10-bit conversion. Attempting to do other than a 10-bit conversion is not recommended.

Since the capacitance of the comparator input is around 0.5pf, the sample and hold circuit must be able to charge a 10pf (20*0.5pf) capacitor without a significant voltage change. To keep the effect of the sample and hold circuit below ± ½ lsb on a 10-bit converter, the voltage on the sample and hold circuit may vary no more than 0.05% (1/2048).

The effective capacitance of the sample and hold must, therefore, be at least 20000pf or 0.02 uf. If there is external leakage on the capacitor, its value must be increased to compensate for the leakage. At 10μA leakage, 2.5 mV (5/2048) will be lost from a 0.17 uf capacitor in 42 μS. The capacitor connected externally to the pin should, therefore, be at least 0.2 uf for best results. If the external signal changes slowly relative to 42 μS, then a larger capacitor will work well and also filter out unwanted noise.

The converter is a 10-bit, successive approximation, ratiometric converter, so the numerical value obtained from the conversion will be:

1023 * (VIN-ANGND) / (VREF-ANGND)

It can be seen that the power supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to a clean ground, as close to the power supply as possible. VREF should be well regulated and used only for the A to D converter. If ratiometric information is derived, VREF can be connected to VCC, but this should be done at the power supply not at the chip. It needs to be able to source around 15 milliamps. Bypass capacitors should be used between VREF and ANGND. ANGND should be within about a tenth of a volt of VSS and VREF should be within a few tenths of a volt of VCC. A 0.01 uf capacitor should be connected between the ANGND and

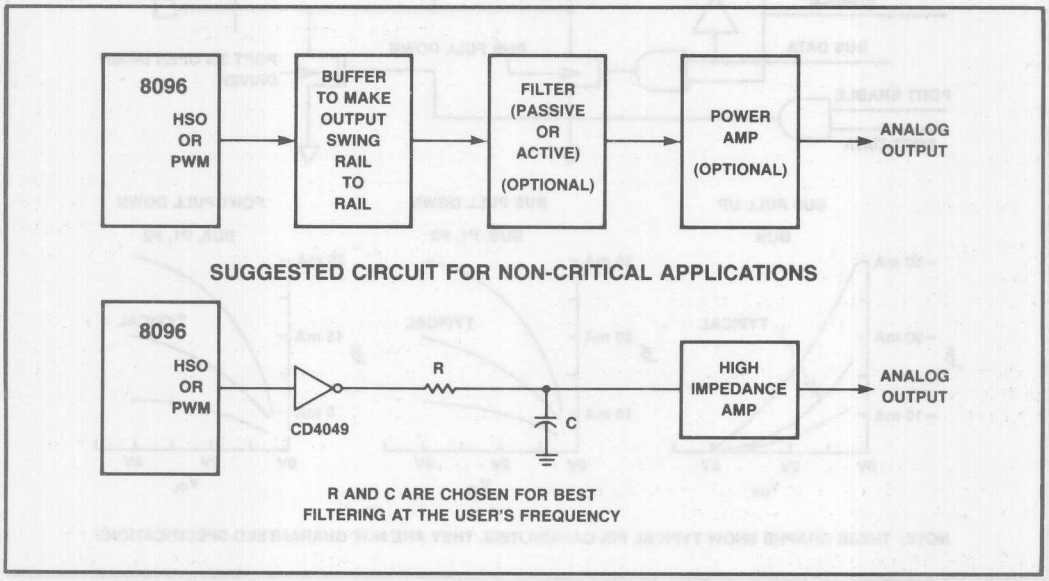


Figure 4-11. D/A Buffer Block Diagram

VBB pins to reduce the noise on VBB and provide the highest possible accuracy. Figure 4-5 shows all of these connections.

4.3.2. Analog Output Suggestions

Analog outputs can be generated by two methods, either by using the PWM output or the HSO. Either device will generate a rectangular pulse training that varies in duty cycle and (for the HSO only) period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to TTL levels. A block diagram of the type of circuit needed is shown in Figure 4-11. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can be used to generate these waveforms if a fixed period on the order of 64 uS is acceptable. If this is not the case then the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 131 milliseconds. Both of these outputs produce TTL levels.

4.4. I/O TIMINGS

The I/O pins on the 8096 are sampled and changed at specific times within an instruction cycle. The timings shown in this section are idealized; no propagation delay factors have been taken into account. Designing a system that depends on an I/O pin to change within a window of less than 50 nanoseconds using the information in this section is not recommended.

4.4.1. HSO Outputs

Changes in the HSO lines are synchronized to Timer 1. All of the HSO lines due to change at a certain value of a timer will change just prior to the incrementing of Timer 1. This corresponds to an internal change during Phase C, every eight state times. From an external perspective the HSO pin should change around the rising edge of CLKOUT and be stable by its falling edge.

Timer 2 is synchronized to increment no faster than Timer 1, so there will always be at least one incrementing of Timer 1 while Timer 2 is at a specific value.

4.4.2. HSI Input Sampling

The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1 full state time to guarantee that it is recognized. This restriction applies even if the divide by eight mode is being used. If two events occur on the same pin within the same 8 state time window, only one of the events will be recorded. The 8 state time window, (ie. the amount of time during which Timer 1 remains constant), is stable to within about 20 nanoseconds. The window starts roughly around the rising edge of CLKOUT, however this timing is very approximate due to the amount of internal circuitry involved.

4.4.3. Standard I/O Port Pins

Port 0 is different from the other digital ports in that it is actually part of the A to D converter. The port is sampled once every 8 state times, the same frequency at which the comparator is charged-up during an A to D conversion. This 8 state times counter is not synchronized with Timer 1. If this port is used the input signal on the pin must be stable 8 state times prior to reading the SFR.

Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from '0' to a '1' the low impedance pull-up will remain on for one state time after the change.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drain, their structure is different when they are used as part of the bus.

4.5. SERIAL PORT TIMINGS

The serial port on the 8096 was designed to be compatible with the 8051 serial port. Since the 8051 uses a divide by 2 clock and the 8096 uses a divide by 3, the serial port on the 8096 had to be provided with its own clock circuit to maximize its compatibility with the 8051 at high baud rates. This means that the serial port itself does not know about state times. There is circuitry which is synchronized to the serial port and to the rest of the 8096 so that information can be passed back and forth.

The baud rate generator is clocked by either XTAL1 or T2CLK, because T2CLK needs to be synchronized to the XTAL1 signal its speed must be limited to 1/16 that of XTAL1. The serial port will not function during the time between the consecutive writes to the baud rate register. Section 2.11.4 discusses programming the baud rate generator.

4.5.1. Mode 0

Mode 0 is the shift register mode. The TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 4-12 shows the waveforms and timing. Note that the port starts functioning when a '1' is written

to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port can be used to expand the I/O capability of the 8096 by simply adding shift registers. A schematic of a typical circuit is shown in Figure 4-13. This circuit inverts the data coming in, so it must be re-inverted in software. The enable and latch connections to the shift registers can be driven by decoders, rather than directly from the low speed I/O ports, if the software and hardware are properly designed.

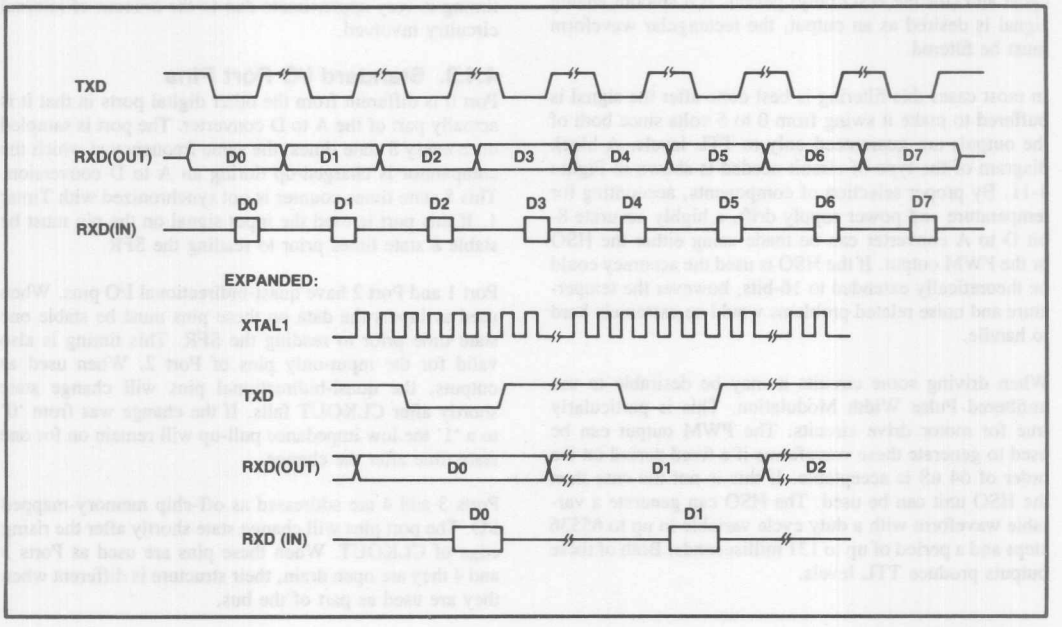


Figure 4-12. Serial Port Timings in Mode 0

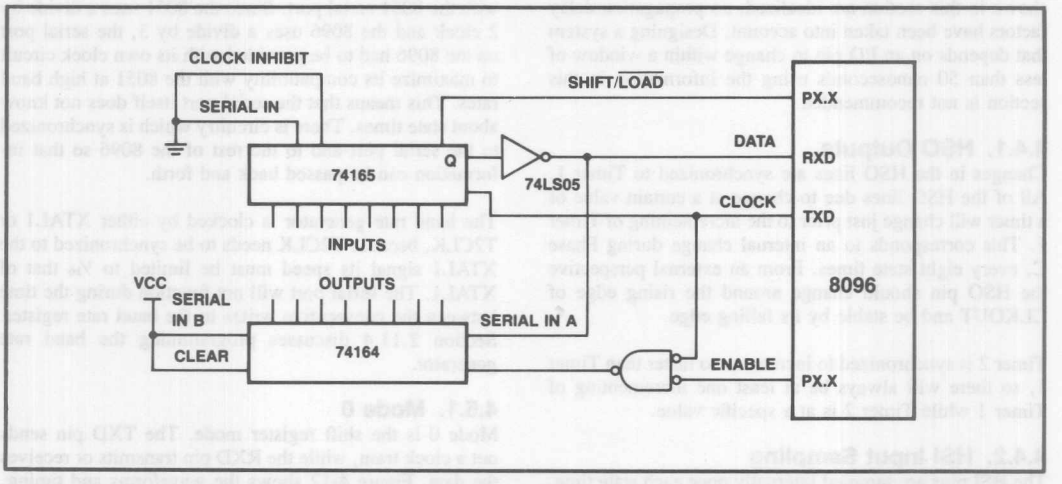


Figure 4-13. Mode 0 Serial Port Example

4.5.2. Mode 1 Timings

Mode 1 operation of the serial port makes use of 10-bit data packages, a start bit, 8 data bits and a stop bit. The transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud rate generator is initialized, the receive shift clock is reset when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

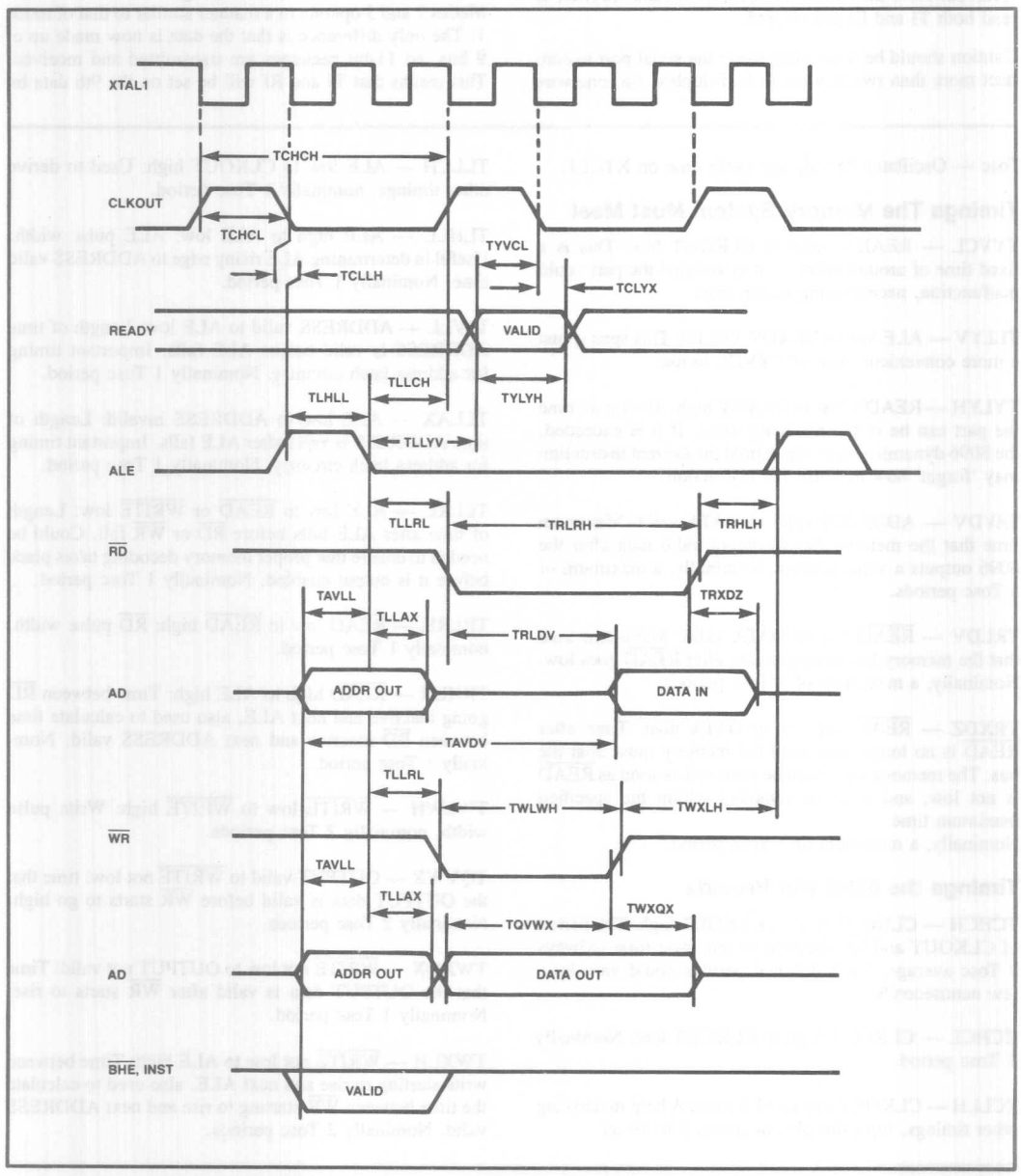


Figure 4-14. Bus Signal Timings

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set when 8 data bits are received, not when the stop bit is received. Note that when the serial port status register is read both TI and RI are cleared.

Caution should be used when using the serial port to connect more than two devices in half-duplex, (ie. one wire for transmit *and* receive). If the receiving processor does not wait for one bit time after RI is set before starting to transmit, the stop bit on the link could be squashed. This could cause a problem for other devices listening on the link.

4.5.3. Mode 2 and 3 Timings

Modes 2 and 3 operate in a manner similar to that of mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit

Tosc — Oscillator Period, one cycle time on XTAL1.

Timings The Memory System Must Meet

TYVCL — READY valid to CLKOUT low: This is a fixed time of around 40 ns, if it is violated the part could malfunction, necessitating a chip reset.

TLLYV — ALE low to READY VALID: This spec is just a more convenient form of TYVCL to use.

TYLYH — READY low to READY high: Maximum time the part can be in the not-ready state. If it is exceeded, the 8096 dynamic nodes which hold the current instruction may 'forget' how to finish the instruction.

TAVDV — ADDRESS valid to DATA valid: Maximum time that the memory has to output valid data after the 8096 outputs a valid address. Nominally, a maximum of 5 Tosc periods.

TRLDV — $\overline{\text{READ}}$ low to DATA valid: Maximum time that the memory has to output data after $\overline{\text{READ}}$ goes low. Nominally, a maximum of 3 Tosc periods.

TRXDZ — $\overline{\text{READ}}$ not low to DATA float: Time after $\overline{\text{READ}}$ is no longer low until the memory must float the bus. The memory signal can be removed as soon as $\overline{\text{READ}}$ is not low, and must be removed within the specified maximum time.
Nominally, a maximum of 1 Tosc period.

Timings the 8096 Will Provide

TCHCH — CLKOUT high to CLKOUT high: The period of CLKOUT and the duration of one state time. Always 3 Tosc average, but individual periods could vary by a few nanoseconds.

TCHCL — CLKOUT high to CLKOUT low: Nominally 1 Tosc period.

TCLLH — CLKOUT low to ALE high: A help in deriving other timings, typically plus or minus 5 to 10 ns.

TLLCH — ALE low to CLKOUT high: Used to derive other timings, nominally 1 Tosc period.

TLHLL — ALE high to ALE low: ALE pulse width. Useful in determining ALE rising edge to ADDRESS valid time. Nominally 1 Tosc period.

TAVLL — ADDRESS valid to ALE low: Length of time ADDRESS is valid before ALE falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLAX — ALE low to ADDRESS invalid: Length of time ADDRESS is valid after ALE falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLRL — ALE low to $\overline{\text{READ}}$ or $\overline{\text{WRITE}}$ low: Length of time after ALE falls before $\overline{\text{RD}}$ or $\overline{\text{WR}}$ fall. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 1 Tosc period.

TRLRH — $\overline{\text{READ}}$ low to $\overline{\text{READ}}$ high: $\overline{\text{RD}}$ pulse width, nominally 1 Tosc period.

TRHLH — $\overline{\text{READ}}$ high to ALE high: Time between $\overline{\text{RD}}$ going inactive and next ALE, also used to calculate time between $\overline{\text{RD}}$ inactive and next ADDRESS valid. Nominally 1 Tosc period.

TWLWH — $\overline{\text{WRITE}}$ low to $\overline{\text{WRITE}}$ high: Write pulse width, nominally 2 Tosc periods.

TQVWX — OUTPUT valid to $\overline{\text{WRITE}}$ not low: time that the OUTPUT data is valid before $\overline{\text{WR}}$ starts to go high. Nominally 2 Tosc periods.

TWXQX — $\overline{\text{WRITE}}$ not low to OUTPUT not valid: Time that the OUTPUT data is valid after $\overline{\text{WR}}$ starts to rise. Nominally 1 Tosc period.

TWXLH — $\overline{\text{WRITE}}$ not low to ALE high: Time between write starting to rise and next ALE, also used to calculate the time between $\overline{\text{WR}}$ starting to rise and next ADDRESS valid. Nominally 2 Tosc periods.

Figure 4-15. Timing Specification Explanations

rather than the 8th. The 9th bit can be used for parity or multiple processor communications (see section 2.11).

4.6. BUS TIMING AND MEMORY INTERFACE

4.6.1. Bus Functionality

The 8096 has a multiplexed (address/data) 16 bit bus. There are control lines provided to demultiplex the bus (ALE), indicate reads or writes ($\overline{RD}$, $\overline{WR}$), indicate if the access is for an instruction (INST), and separate the bus into high and low bytes ($\overline{BHE}$, AD0). Section 2.3.5 contains an overview of the bus operation.

4.6.2. Timing Specifications

Figure 4-14 shows the timing of the bus signals and data lines. Since this is a new part, the exact timing specifications are subject to change, please refer to the latest 8096 data sheet to ensure that your system is designed to the proper specifications. The major timing specifications are described in Figure 4-15.

4.6.3. READY Line Usage

When the processor has to address a memory location that cannot respond within the standard specifications it is necessary to use the READY line to generate wait states. When the READY line is held low the processor waits in a loop for the line to come high. There is a maximum time that the READY line can be held low without risking a processor malfunction due to dynamic nodes that have not been refreshed during the wait states. This time is shown as TYLYH in the data sheet.

In most cases the READY line is brought low after the address is decoded and it is determined that a wait state is needed. It is very likely that some addresses, such as those addressing memory mapped peripherals, would need wait states, and others would not. The READY line must be stable within the TLLYV specification after ALE falls (or the TYVCL before CLKOUT falls) or the processor could lock-up. There is no requirement as to when READY may go high, as long as the maximum READY low time (TYLYH) is not violated.

4.6.4. INST Line Usage

The INST (Instruction) line is high during the output of an address that is for an instruction stream fetch. It is low during the same time for any other memory access. At any other time it is not valid. This pin is not present on the 48-pin versions. The INST signal can be used with a logic analyzer to debug a system. In this way it is possible to determine if the fetch was for instructions or data, making the task of tracing the program much easier.

4.6.5. Address Decoding

The multiplexed bus of the 8096 must be demultiplexed before it can be used. This ca be done with 2 74LS373 transparent latches. As explained in section 2.3.5, the latched address signal will be referred to as MA0 through MA15. (Memory Address), and the data lines will be called MD0 through MD15, (Memory Data).

Since the 8096 can make accesses to memory for either bytes or words it is necessary to have a way of determining the type of access desired. The $\overline{BHE}$ and MA0 lines are used for this purpose. $\overline{BHE}$ must be latched, as it is valid only when the address is valid. The memory system is typically set up as 32K by 16, instead of 64K by 8. When the $\overline{BHE}$ line is low, the upper byte is enabled. When MA0 is low, the lower byte is enabled when MA0 is high $\overline{BHE}$ will be low, and the upper byte is enabled.

When external RAM and EPROM are both used in the system the control logic can be simplified a little to some of the addresses. The 8096 will always output BHE to indicate if a read is of the high byte or the low byte, but it discards the byte it is not going to use. It is therefore possible to use the $\overline{BHE}$ and MA0 lines only to control

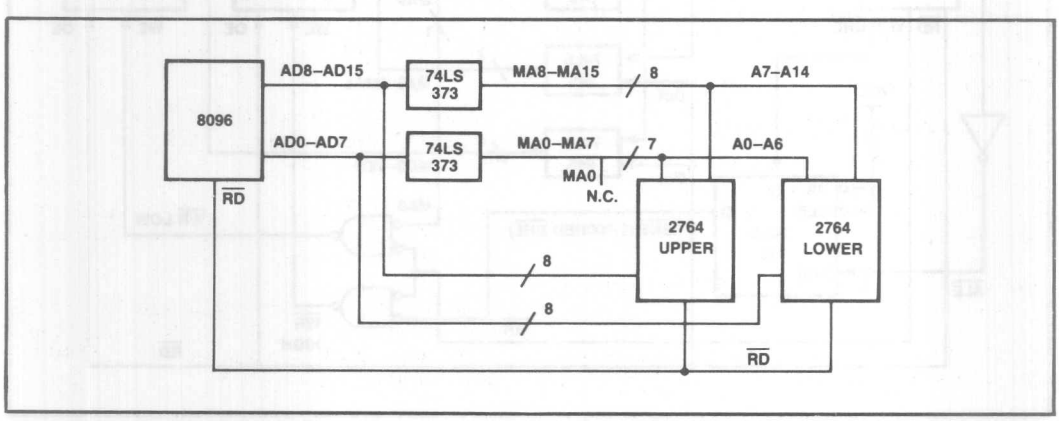


Figure 4-16. Memory Wiring Example–EPROM Only System

a RAM/ROM system.

4.6.6. System Verification Example

To verify that a system such as the one in Figure 4-17 will work with the 8096, it is necessary to check all of the timing parameters. Let us examine this system one parameter at a time using the proposed 8096 specifications. These specifications are subject to change, refer to the latest 8096 data sheet for the current specifications.

The timings of signals that the processor and memory use are effected by the latch and buffer circuitry. The timings of the signal provided by the processor are delayed by various amounts of time. Similarly, the signals coming back from the memory are also delayed. The calculations involved in verifying this system follow:

Address Valid Delay — 20 nanoseconds

The address lines are delayed by passing them through the 74LS373s, this delay is specified at 18ns after Address is valid or 30ns after ALE is high. Since the

If Limited by ALE:

Minimum ALE pulse width = Tosc-10 (TLHLL)

Minimum Addr set-up to ALE falling = Tosc-20 (TAVLL)

Therefore ALE could occur 10 ns after Address valid.

Total delay from 8096 Address stable to MA (Memory Address) stable would be:

ALE delay from address	−10
74LS373 clock to output	30
	20 nanoseconds

If Limited by Address Valid:

74LS373 Data Valid to Data Output = 18 nanoseconds

In the worst case, the delay in Address valid is controlled by ALE and has a value of 20 nanoseconds.

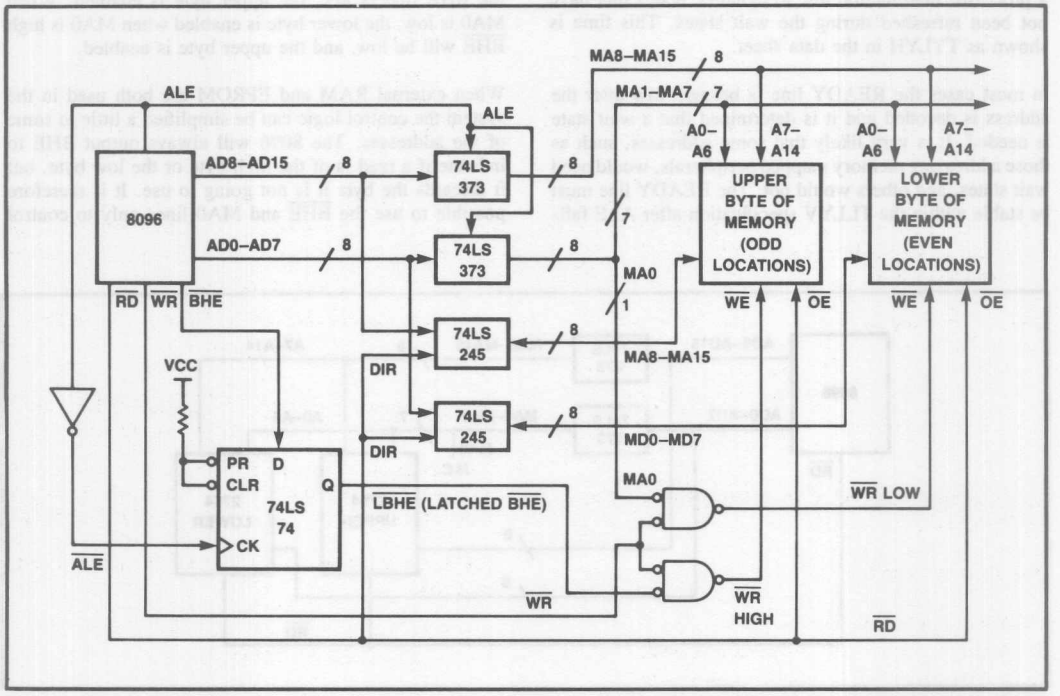


Figure 4-17. RAM/ROM Memory System

Delay of Data Transfer to/from Processor — 12 nanoseconds

The $\overline{RD}$ low to Data valid specification (TRLDV) is 3 Tosc-50, (200 ns at 12 MHz). The 74LS245 is enabled by $\overline{RD}$ and has a delay of 40 ns from enable. The enable delay is clearly not a problem.

The 74LS245 is enabled except during a read, so there is no enable delay to consider for write operaions.

The Data In to Data Out delay of the 74LS245 is 12 ns.

Delay of $\overline{WR}$ signal to memory — 15 nanoseconds

Latched $\overline{BHE}$ is delayed by the inverter on ALE and the 74LS74.

74LS04 delay (Output low to high)	= 22
74LS74 delay (Clock to Output)	= 40
Delay of Latched $\overline{BHE}$ from ALE falling	= 62 nanoseconds

The 74LS74 requires data valid for 20 ns prior to the clock, the 8096 will have $\overline{BHE}$ stable Tosc-20 ns (TAVLL, 63 ns at 12 MHz) prior to ALE falling. There is no problem here.

MA0 is valid prior to ALE falling, since the 20 ns Address Delay is less than TAVLL.

$\overline{WR}$ will fall no sooner than Tosc-20 ns (TLLRL, 63 ns at 12 MHz) after ALE goes low. It will therefore be valid just after the Latched $\overline{BHE}$ is valid, so it is the controlling signal.

$\overline{WR}$ High and $\overline{WR}$ Low are valid 15 ns after MA0, Latched $\overline{BHE}$ and $\overline{WR}$ are valid. Since $\overline{WR}$ is the last signal to go valid, the delay of $\overline{WR}$ (High and Low) to memory is 15 ns.

Delay Summary —

Address Delay	= 20 ns
Data Delay	= 12 ns
$\overline{WR}$ Delay	= 15 ns
$\overline{RD}$ Delay	= 0 ns

Characteristics of a 12 MHz 8096 system with latches:

Required by system:

Address valid to Data in;

TAVDV	:	386.6 ns maximum
Address Delay	: −	20.0 ns maximum
Data Delay	: −	12.0 ns maximum
		354.6 ns maximum

Read low to Data in;

TRLDV	:	200.0 ns maximum
$\overline{RD}$ Delay	: −	00.0 ns maximum
Data Delay	: −	12.0 ns maximum
		188.0 ns maximum

Provided by System:

Address valid to Control;

TLLRL	:	63.3 ns minimum
TAVLL	:	63.3 ns minimum
Address Delay	: −	20.0 ns maximum
$\overline{WR}$ Delay	: −	00.0 ns minimum (no spec)
		101.6 ns minimum

Write Pulse Width;

TWLWH	:	151.6 ns minimum
Rising $\overline{WR}$ Delay	: −	15.0 ns maximum
Falling $\overline{WR}$ Delay	:	00.0 ns minimum (no spec)
		146.6 ns minimum

Data Setup to $\overline{WR}$ rising;

TQVWX	:	136.6 ns minimum
Data Delay	: −	12.0 ns maximum
$\overline{WR}$ Delay	:	00.0 ns minimum (no spec)
		124.6 ns minimum

Data Hold after $\overline{WR}$;

TWXQX	:	58.3 ns minimum
Data Delay	:	0.0 ns minimum (no spec)
$\overline{WR}$ Delay	: −	15.0 ns maximum
		43.3 ns minimum

The two memory devices which are expected to be used most often with the 8096 are the 2764 EPROM and the 2128 RAM. The system verification for the 2764 is simple.

2764 Tac

(Address valid to Output) < Address valid to Data in
250 ns < 354 ns O.K.

2764 Toe

(Output Enable to Output) < $\overline{Read}$ low to Data in
100 ns < 188 ns O.K.

These calculations assume no address decoder delays and no delays on the $\overline{RD}$ (OE) line. If there are delays in these signals the delays must be added to the 2764's timing.

The read calculations for the 2128 are similar to those for the 2764.

2128-20 Tac < Address valid to Data in
200 ns < 354 ns O.K.

2128-20 Toe < $\overline{Read}$ low to Data in
65 ns < 188 ns O.K.

The write calculations are a little more involved, but still straight-forward.

2128 Twp (Write Pulse) < Write Pulse Width
100 ns < 146 ns O.K.

2128 Tds (Data Setup) < Data Setup to $\overline{WR}$ rising
65 ns < 124 ns O.K.

2128 Tdh (Data Hold) < Data Hold after $\overline{WR}$
0 ns < 43 ns

All of the above calculations have been done assuming that no components are in the circuit except for those shown in Figure 4-17. If additional components are added, as may be needed for address decoding or memory bank switching, the calculations must be updated to reflect the actual circuit.

4.6.7. I/O Port Reconstruction

When a single-ship system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O ports 3 and 4 using a memory-mapped I/O technique. The circuit shown in Figure 4-18 provides this function on the iSBE-96 emulator board. It can be attached to any 8096 system which has the required address decoding and bus demultiplexing.

The output circuitry is basically just a latch that operates when 1FFEH or 1FFFH are placed on the MA lines. The inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 8096. The 'reset' line is used to set the ports to all 1's when the 8096 is reset. It should be noted that the voltage and current characteristics of this port will differ from those of the 8096, but the basic functionality will be the same.

The input circuitry is just a bus transceiver that is addressed at 1FFEH or 1FFFH. If the ports are going to be used for either input or output, but not both, some of the circuitry can be eliminated.

4.7. NOISE PROTECTION TIPS

Designing controllers differs from designing other computer equipment in the area of noise protection. A microcontroller circuit under the hood of a car, in a photocopier, CRT terminal, or a high speed printer is subject

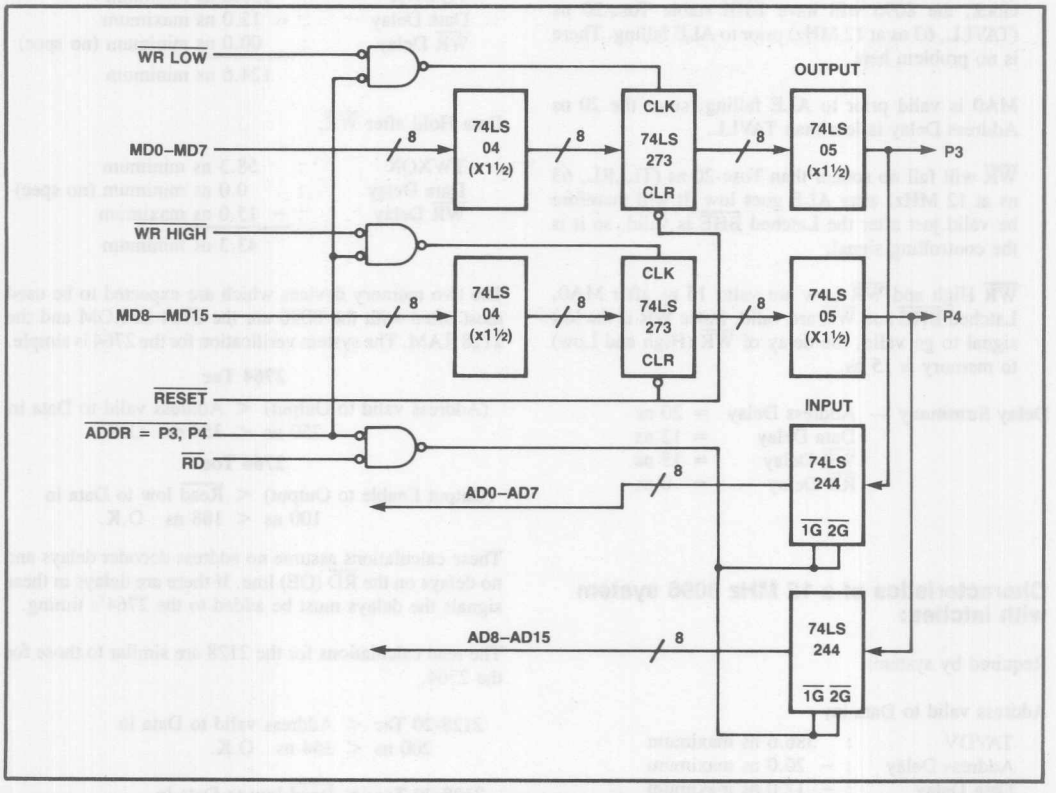


Figure 4-18. I/O Port Reconstruction

to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the pc board to find itself in the path of electrostatic discharges. Glitches and noise on the pc board can cause the processor to act unpredictably, usually by changing either the memory locations or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8096 has a watchdog timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096 instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

Many hardware solutions exist for keeping pc board noise to a minimum. Ground planes, gridded ground and VCC structures, bypass capacitors, transient absorbers and power busses with built-in capacitors can all be of great help. It is much easier to design a board with these features than to try to retrofit them later. Proper pc board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Intel Application Note AP-125, "Designing Microcontroller Systems For Noisy Environments."

4.8. PACKAGING PINOUTS AND ENVIRONMENT

The MCS-96 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, with or without ROM, and with or without an A to D converter. A summary of the available options is shown in Figure 4-19.

The 48-pin versions are available in a 48-pin DIP (Dual In-Line) package, in either ceramic or plastic.

The 68-pin versions are available in a ceramic pin grid array, and a plastic flatpack. A plastic pin grid array will be available in the near future.

	ROMLESS		WITH ROM	
	68-pin	48-pin	68-pin	48-pin
Without A to D	8096	8094	8396	8394
With A to D	8097	8095	8397	8395

Figure 4-19. The MCS®-96 Family of Products

MCS®-96 Data Sheets

5

8094/8095/8096/8097
8394/8395/8396/8397
16-BIT MICROCONTROLLERS

- **839X: an 809X with 8K Bytes of On-chip ROM**
- **High Speed Pulse I/O**
- **10-bit A/D Converter**
- **8 Interrupt Sources**
- **Pulse-Width Modulated Output**
- **Four 16-bit Software Timers**
- **232 Byte Register File**
- **Memory-to-Memory Architecture**
- **Full Duplex Serial Port**
- **Five 8-bit I/O Ports**
- **Watchdog Timer**

The MCS-96 family of 16-bit microcontrollers consists of 8 members, all of which are designed for high-speed control functions.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0 μsec and a 16 x 16-bit multiply or 32/16-bit divide in 6.5 μsec. Instruction execution times average 1 to 2 μsec in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

An on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values. This feature is only available on the 8095, 8395, 8097 and 8397.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.

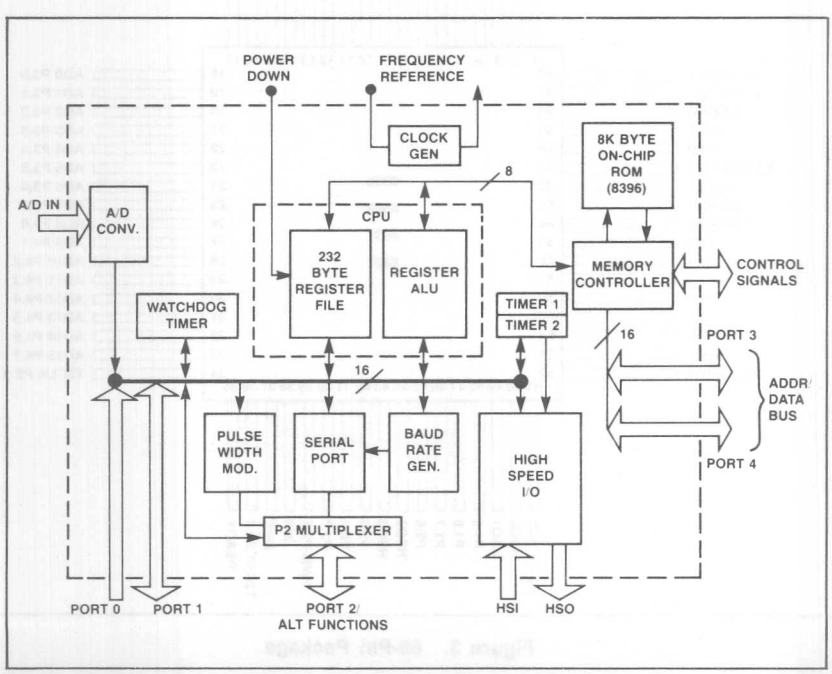


Figure 1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)

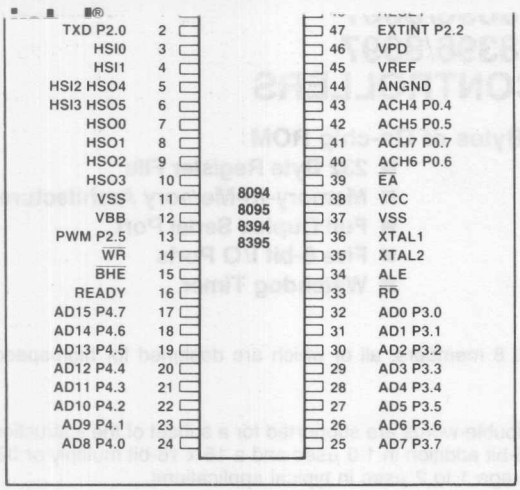


Figure 2. 48-Pin Package

generally referred to as the 8096. The 8096 is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM. The MCS-96 numbering system is shown below:

OPTIONS		68 PIN	48 PIN
DIGITAL I/O	ROMLESS	8096	8094
	ROM	8396	8394
ANALOG AND DIGITAL I/O	ROMLESS	8097	8095
	ROM	8397	8395

Figures 2 and 3 show the pinouts for the 48- and 68-pin packages.

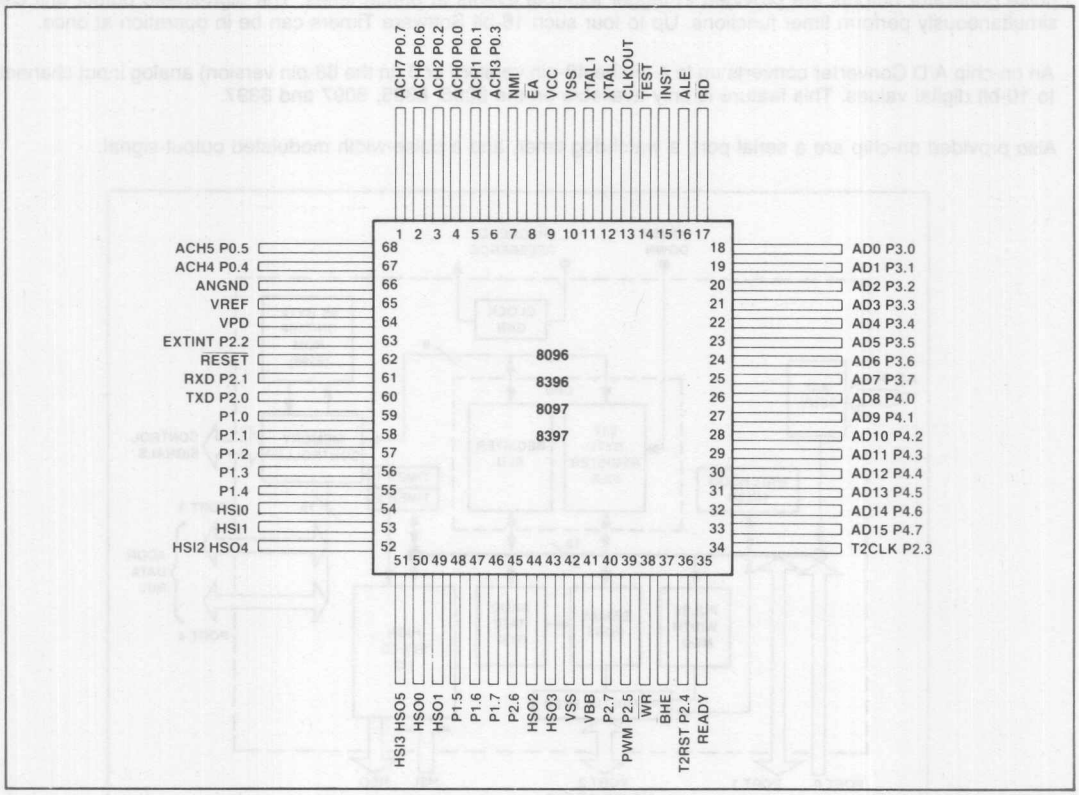


Figure 3. 68-Pin Package

Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D+A	√	√	√	√	√	—	
ADD/ADDB	3	D ← B+A	√	√	√	√	√	—	
ADDC/ADDCB	2	D ← D+A+C	√	√	√	√	√	—	
SUB/SUBB	2	D ← D − A	√	√	√	√	√	—	
SUB/SUBB	3	D ← B − A	√	√	√	√	√	—	
SUBC/SUBCB	2	D ← D − A + C − 1	√	√	√	√	√	—	
CMP/CMPB	2	D − A	√	√	√	√	√	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	√	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	√	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	√	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	√	3
DIV/DIVU	2	D ← (D, D + 2)/A D + 2 remainder	—	—	—	√	√	—	2
DIVB/DIVUB	3	D ← (D, D + 1)/A D + 1 remainder	—	—	—	√	√	—	3
AND/ANDB	2	D ← D and A	√	√	0	0	—	—	
AND/ANDB	3	D ← B and A	√	√	0	0	—	—	
OR/ORB	2	D ← D or A	√	√	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	√	√	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3,4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3,4
PUSH	1	SP ← SP − 2; (SP) A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2	√	√	√	√	√	√	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
INDJMP	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J(conditional)	1	PC ← PC + 8-bit offset	—	—	—	—	—	—	5
JC		Jump if C = 1	—	—	—	—	—	—	5
JNC		Jump if C = 0	—	—	—	—	—	—	5

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (continued)

Mnemonic	Oper-ands	Operation (Note 1)	Flags Z	N	C	V	VT	ST	Notes
JE		Jump if Z = 1	—	—	—	—	—	—	5
JNE		Jump if Z = 0	—	—	—	—	—	—	5
JGE		Jump if N = 0	—	—	—	—	—	—	5
JLT		Jump if N = 1	—	—	—	—	—	—	5
JGT		Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE		Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH		Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH		Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV		Jump if V = 1	—	—	—	—	—	—	5
JNV		Jump if V = 0	—	—	—	—	—	—	5
JVT		Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT		Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST		Jump if ST = 1	—	—	—	—	—	—	5
JNST		Jump if ST = 0	—	—	—	—	—	—	5
JBS		Jump if ST = 1	—	—	—	—	—	—	5,6
JBC		Jump if Specified Bit = 0	—	—	—	—	—	—	5,6
DJNZ	1	D ← D − 1; if D ≠ then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	√	√	√	√	√	—	
NEG/NEGB	1	D ← 0 − D	√	√	√	√	√	—	
INC/INCB	1	D ← D + 1	√	√	√	√	√	—	
EXT	1	D ← D; D + 2 ← Sign (D)	√	√	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	√	√	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	√	√	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	1	C ← msb — — — — — lsb ← 0	√	√	√	√	√	—	7
SHR/SHRB/SHRL	1	0 → msb — — — — — lsb → C	√	√	√	0	—	√	7
SHRA/SHRAB/SHRAL	1	msb → msb — — — — — lsb → C	√	√	√	0	—	√	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts	—	—	—	—	—	—	
EI	0	Enable All Interrupts	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Normalize	√	1	—	—	—	—	7

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.

Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊛ NORMAL			INDIRECT⊛ AUTO-INC.		INDEXED⊛ SHORT			INDEXED⊛ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	27/32	3	28/33	8F	4	27/32	5	28/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	19/24	3	20/25	9F	4	19/24	5	20/25
DIV	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
DIVB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29

Notes:

⊛ Long indexed and Indirect + instructions have identical opocodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

Opcode and State Time Listing (Continued)

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⑧ NORMAL			INDIRECT⑧ AUTO-INC.		INDEXED⑧ SHORT			INDEXED⑧ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	——	C2	3	7/13	3	8/14	C3	4	7/13	5	8/14
STB	2	C4	3	4	—	—	——	C6	3	7/13	3	8/14	C7	4	7/13	5	8/14
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	——	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	——	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
INDJMP	E3	2	8	RET	F0	1	12/16⑤

Notes:

① Number of state times shown for internal/external operands.

③ This instruction is generated by the assembler when a branch indirect (BR [Rx]) instruction is reached.

④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.

⑤ State times for stack located internal/external.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

DJNZ OPCODE E0; 3 BYTES; 5/9 STATE TIMES (NOT TAKEN/TAKEN)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR	WORD		INSTR	BYTE		INSTR	DBL WD		
MNEMONIC	OP	B	MNEMONIC	OP	B	MNEMONIC	OP	B	STATE TIMES
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT⑦
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT⑦
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT⑦

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST	FF	1	16	SKIP	00	2	4

NORMALIZE

NORML	0F	3	11 + 1 PER SHIFT

Notes:

⑥ This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.

⑦ Execution will take at least 8 states, even for 0 shift.

ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias . . . . . 0°C to +70°C
Storage Temperature . . . . . -40°C to +150°C
Voltage from Any Pin to VSS or ANGND . . . -0.3V to +7.0V
Average Output Current from Any Pin . . . . . 10 mA
Power Dissipation . . . . . 1.5 Watts

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
TA	Ambient Temperature Under Bias	0	+70	C
VCC	Digital Supply Voltage	4.50	5.50	V
VREF	Analog Supply Voltage	4.5 VCC−0.3	5.5 VCC+0.3	V V
fOSC	Oscillator Frequency	6.0	12	MHz
VPD	Power-Down Supply Voltage	4.50	5.50	V

VBB should be connected to ANGND through a 0.01 μF capacitor. ANGND and VSS should be nominally at the same potential.

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Units	Test Conditions
VIL	Input Low Voltage	−0.3	+0.8	V	
VIH	Input High Voltage (Except $\overline{\text{RESET}}$)	2.0	VCC+0.5	V	
VIH1	Input High Voltage, $\overline{\text{RESET}}$ Rising	2.4	VCC+0.5	V	
VIH2	Input High Voltage, $\overline{\text{RESET}}$ Falling	2.0	VCC+0.5	V	
VOL	Output Low Voltage		0.45	V	See Note 1.
VOH	Output High Voltage	2.4		V	See Note 2.
ICC	VCC Supply Current		200	mA	All outputs disconnected.
IPD	VPD Supply Current		1	mA	Normal operation and Power-Down.
IREF	VREF Supply Current		15	mA	
ILI	Input Leakage Current to all pins of HSI, P0, P3, P4, and to P2.1.		±10	μA	Vin=0 to VCC See Note 3
IIH	Input High Current to $\overline{\text{EA}}$		100	μA	VIH=2.4V
IIL	Input Low Current to all pins of P1, and to P2.6, P2.7.		−100	μA	VIL=0.45V
IIL1	Input Low Current to $\overline{\text{RESET}}$		−2	mA	VIL=0.45V
IIL2	Input Low Current P2.2, P2.3, P2.4, READY		−50	μA	VIL=0.45V
Cs	Pin Capacitance (Any Pin to VSS)		10	pF	fTEST=1MHz

NOTES:

1. IOL = 0.36 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports.
   IOL = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{\text{BHE}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).

2. IOH = −20 μA for all pins of P1, for P2.6 ad P2.7.
   IOH = −200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{\text{BHE}}$, $\overline{\text{WR}}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).
   P3 and P4, when used as ports, have open-drain outputs.

3. Analog Conversion not in process.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at VREF = 5.120 volts.

Resolution ..... ±0.001 VREF
Accuracy ..... ±0.004 VREF
Differential nonlinearity ..... ±0.002 VREF max
Integral nonlinearity ..... ±0.004 VREF max
Channel-to-chanel matching ..... ±1 LSB
Crosstalk (DC to 100kHz) ..... −60dB max

AC CHARACTERISTICS

Test Conditions; Oscillator Frequency = 12MHz
Load Capacitance on Output Pins = 80pF
Other Operating Conditions as previously described.

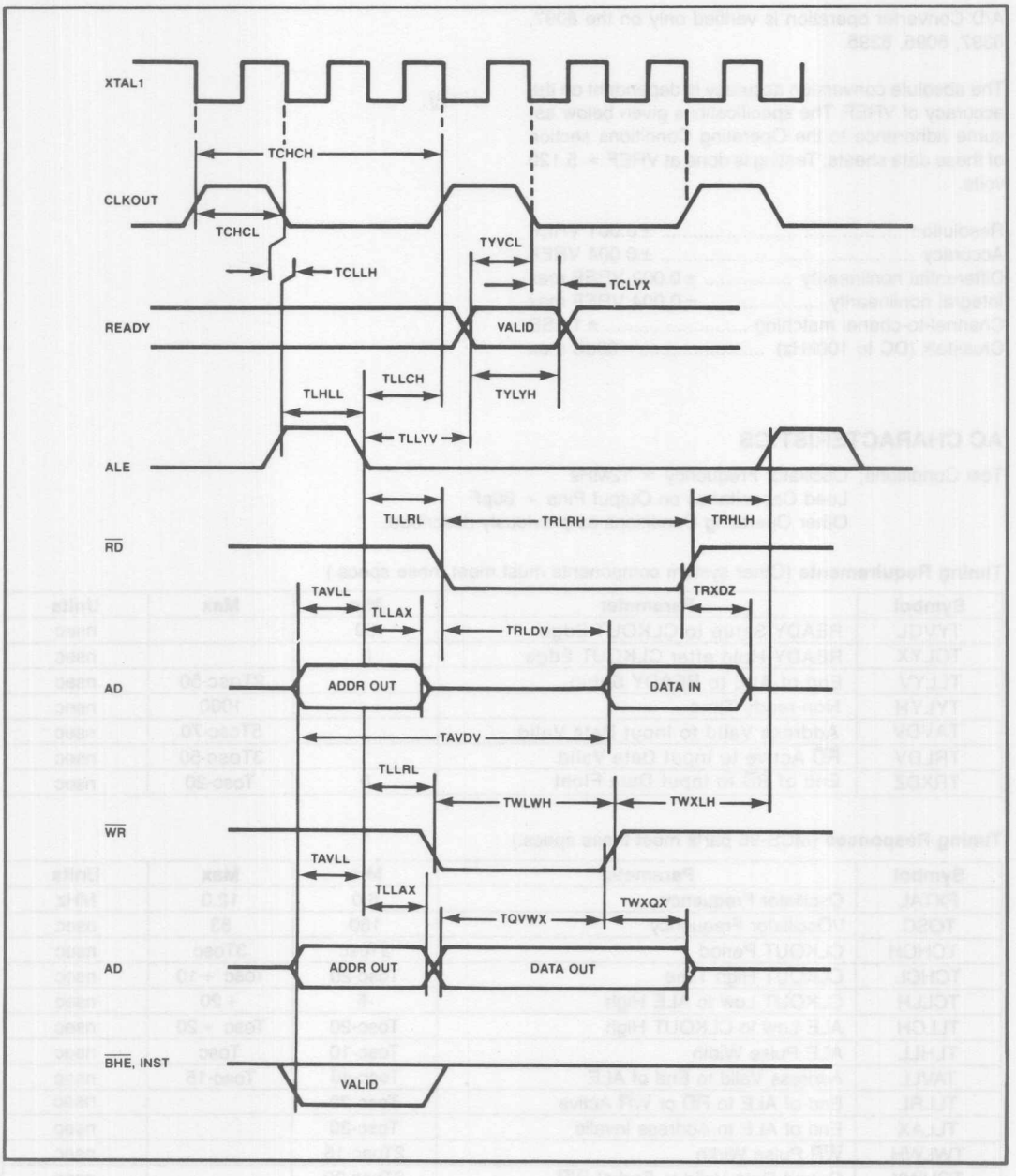
Timing Requirements (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
TYVCL	READY Setup to CLKOUT Edge	50		nsec
TCLYX	READY Hold after CLKOUT Edge	0		nsec
TLLYV	End of ALE to READY Setup		2Tosc-50	nsec
TYLYH	Non-ready Time		1000	nsec
TAVDV	Address Valid to Input Data Valid		5Tosc-70	nsec
TRLDV	$\overline{RD}$ Active to Input Data Valid		3Tosc-50	nsec
TRXDZ	End of $\overline{RD}$ to Input Data Float	0	Tosc-20	nsec

Timing Responses (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.0	12.0	MHz
TOSC	1/Oscillator Frequency	160	83	nsec
TCHCH	CLKOUT Period	3Tosc	3Tosc	nsec
TCHCL	CLKOUT High Time	Tosc-20	Tosc +10	nsec
TCLLH	CLKOUT Low to ALE High	-5	+20	nsec
TLLCH	ALE Low to CLKOUT High	Tosc-20	Tosc +20	nsec
TLHLL	ALE Pulse Width	Tosc-10	Tosc	nsec
TAVLL	Address Valid to End of ALE	Tosc-40	Tosc-15	nsec
TLLRL	End of ALE to $\overline{RD}$ or $\overline{WR}$ Active	Tosc-20		nsec
TLLAX	End of ALE to Address Invalid	Tosc-20		nsec
TWLWH	$\overline{WR}$ Pulse Width	2Tosc-15		nsec
TQVWX	Output Data Valid to End of $\overline{WR}$	2Tosc-30		nsec
TWXQX	Output Data Hold after $\overline{WR}$	Tosc-25		nsec
TWXLH	End of $\overline{WR}$ to Next ALE	2Tosc-30		nsec
TRLRH	$\overline{RD}$ Pulse Width	3Tosc-30		nsec
TRHLH	End of $\overline{RD}$ to Next ALE	Tosc-25		nsec

WAVEFORM



Bus Signal Timings

INDEX